

# Getting started with XMC7000 MCU on ModusToolbox™ software

## About this document

### Scope and purpose

This application note helps you explore the XMC7000 MCU architecture and development tools and shows you how to create your first project using the Eclipse IDE for ModusToolbox™ software. This application note also guides you to more resources available online to accelerate your learning about XMC7000 MCU.

### Intended audience

This document is intended for the users who are new to XMC7000 MCU and ModusToolbox™ software.

### Associated part family

All [XMC7000 MCU](#) devices.

### Software version

[ModusToolbox™ software](#) 3.2 or above.

### More code examples? We heard you

To access an ever-growing list of XMC7000 code examples using ModusToolbox™, please visit the [GitHub](#) site.

## Table of contents

## Table of contents

	<b>About this document</b> .....	1
	<b>Table of contents</b> .....	2
<b>1</b>	<b>Introduction</b> .....	4
<b>2</b>	<b>Development ecosystem</b> .....	6
2.1	XMC7000 resources .....	6
2.2	Firmware/application development .....	6
2.2.1	Installing the ModusToolbox™ tools package .....	6
2.2.2	Choosing an IDE .....	6
2.2.3	ModusToolbox™ software .....	7
2.2.4	ModusToolbox™ applications .....	9
2.2.5	XMC7000 software resources .....	11
2.2.5.1	Configurators .....	11
2.2.5.2	Library management for XMC7000 MCU .....	12
2.2.5.3	Software development for XMC7000 MCU .....	12
2.3	Support for other IDEs .....	14
2.4	FreeRTOS support with ModusToolbox™ .....	15
2.5	Programming/debugging using Eclipse IDE .....	15
2.6	XMC7000 MCU development kits .....	15
<b>3</b>	<b>Device features</b> .....	16
<b>4</b>	<b>Getting started with XMC7000 MCU design</b> .....	19
4.1	Prerequisites .....	19
4.1.1	Hardware .....	19
4.1.2	Software .....	19
4.2	Using these instructions .....	19
4.3	About the design .....	19
4.4	Create a new application .....	20
4.5	View and modify the design .....	23
4.5.1	Open the Device Configurator .....	25
4.5.2	Add retarget-io middleware .....	26
4.5.3	Configuration of UART, timer peripherals, pins, and system clocks .....	27
4.6	Write firmware .....	27
4.7	Build the application .....	34
4.8	Program the device .....	35
4.9	Test your design .....	37
<b>5</b>	<b>Adding FreeRTOS support for XMC7000</b> .....	40
5.1	For Single-core XMC7000 project .....	40
5.2	For Multi-core project .....	44



Table of contents

---

6	Summary .....	57
7	References .....	58
	Glossary .....	59
	Revision history .....	60
	Disclaimer .....	61

## 1 Introduction

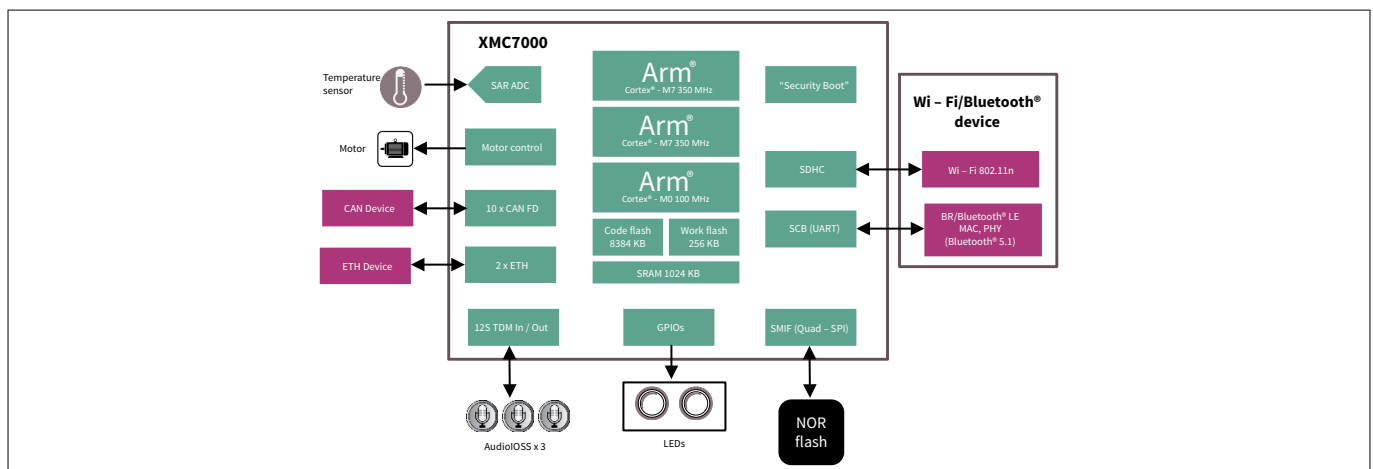
### 1 Introduction

The XMC7000 device is a microcontroller targeted at industrial applications. The XMC7000 integrates the following features on a single chip:

- Up to two 350-MHz 32-bit Arm® Cortex®-M7 CPUs each with:
  - Single-cycle multiply
  - Single-double-precision floating point unit (FPU)
  - 16-KB data cache, 16-KB instruction cache
  - Memory protection unit (MPU)
  - 16-KB instruction and 16-KB data tightly-coupled memory (TCM)
- 100-MHz 32-bit Arm® Cortex® M0+ CPU with single-cycle multiply and MPU
- Programmable analog and digital peripherals
- Up to 8384 KB of code flash with an additional 256 KB of work flash, and an internal SRAM of up to 1024 KB
- XMC7000 MCU is suitable for a variety of power-sensitive applications such as:
  - Wireless charging
  - Lighting
  - Power supply servers
  - Robots and drones
  - MHA
  - Industrial drives
  - PLC
  - I/O modules
  - Electric two-wheelers

The [ModusToolbox™ software environment](#) supports XMC7000 MCU application development with a set of tools for configuring the device, setting up peripherals, and complementing your projects with world-class middleware. See the Infineon GitHub repos for BSPs (Board Support Packages) for all kits, libraries for popular functionality like CAPSENSE™ and emWin, and a comprehensive array of example applications to get you started.

Figure 1 illustrates an application-level block diagram for a real-world use case using XMC7000 MCU.



**Figure 1** Application-level block diagram using XMC7000 MCU

## 1 Introduction

XMC7000 MCU is a highly capable and flexible solution. For example, the real-world use case in [Figure 1](#) takes advantage of these features:

- A buck converter for ultra-low-power operation
- An analog front end (AFE) within the device to condition and measure sensor outputs such as the ambient light sensor
- Serial Communication Blocks (SCBs) to interface with multiple digital sensors such as motion sensors
- Programmable digital logic (smart I/O) and peripherals (Timer Counter PWM or TCPWM) to drive the motor and LEDs respectively
- Up to 10 CAN FD channels with increased data rate (up to 8 Mbps) supports all the requirement of CAN FD specification V1.0 for non-ISO CAN FD
- Up to two 10/100/1000 Mbps Ethernet MAC interfaces conforming to IEEE-802.3az supports MII/RMII/RGMII/AVB/PTP PHY interfaces
- SDIO interface to a Wi-Fi/Bluetooth® device to provide IoT cloud connectivity
- Product security features managed by CM0+ CPU and application features executed by CM7 CPUs

There are two product lines in XMC7000. [Table 1](#) provides overview of different product lines:

**Table 1** XMC7000 MCU product lines

Device series	Details
XMC7100	Triple-core architecture: 250-MHz Arm® Cortex®-M7 and 100-MHz Cortex®-M0+ 4-MB flash, 768-KB RAM Packages: 100/144/176 TEQFP, 272 BGA
XMC7200	Triple-core architecture: 350-MHz Arm® Cortex®-M7 and 100-MHz Cortex®-M0+ 8-MB flash, 1-MB RAM Packages: 176 TEQFP, 272 BGA

**Note:** All features are not available in all devices in a product line. See the [device datasheets](#) for more details.

This application note introduces you to the capabilities of the XMC7000 MCU, gives an overview of the development ecosystem, and gets you started with a simple 'Hello World' application wherein you learn to use the XMC7000 MCU. We will show you how to create an application from an empty starter application, but the completed design is available as a [code example for ModusToolbox™ on GitHub](#).

For hardware design considerations, see [Hardware design guide for the XMC7000 Family](#).

## 2 Development ecosystem

## 2 Development ecosystem

### 2.1 XMC7000 resources

A wealth of data available [here](#) helps you to select the right XMC™ device and quickly and effectively integrate it into your design. For a comprehensive list of XMC7000 MCU resources, see How to design with XMC7000 MCU. The following is an abbreviated list of resources for XMC7000 MCU.

- **Overview:** [XMC7000 MCU webpage](#)
- **Product selectors:** [XMC7000 MCU](#)
- [Datasheets](#) describe and provide electrical specifications for each device family.
- [Application notes](#) and [code examples](#) cover a broad range of topics, from basic to advanced level. You can also browse our [collection of code examples](#).
- [Technical reference manuals \(TRMs\)](#) provide detailed descriptions of the architecture and registers in each device family.
- [Debug XMC7000 MCU in ModusToolbox™ environment](#) provides the information necessary to debug with single-core and multi-core applications.
- **Development tools:** Many low-cost [kits](#) are available for evaluation, design, and development of different applications using XMC7000 MCUs.
- **Technical Support:** [XMC7000 community forum](#), [knowledge base articles](#)

### 2.2 Firmware/application development

There is one development platforms that you can use for application development with XMC7000 MCU:

- **ModusToolbox™:** This software includes configuration tools, low-level drivers, middleware libraries, and other packages that enable you to create MCU and wireless applications. All tools run on Windows, macOS, and Linux. ModusToolbox™ includes an Eclipse IDE, which provides an integrated flow with all the ModusToolbox™ tools. Other IDEs such as Visual Studio Code, IAR Embedded Workbench and Arm® MDK (μVision) are also supported.
  - ModusToolbox™ software supports stand-alone device and middleware configurators. Use the configurators to set the configuration of different blocks in the device and generate code that can be used in firmware development. It is recommended that you use ModusToolbox™ software for all application development for XMC7000 MCUs. See the [ModusToolbox™ tools package user guide](#) for more information.
  - Libraries and enablement software are available at the [GitHub](#) site.
  - ModusToolbox™ tools and resources can also be used in the command line. See the “Using command-line” section in the [ModusToolbox™ user guide](#) for detailed documentation.

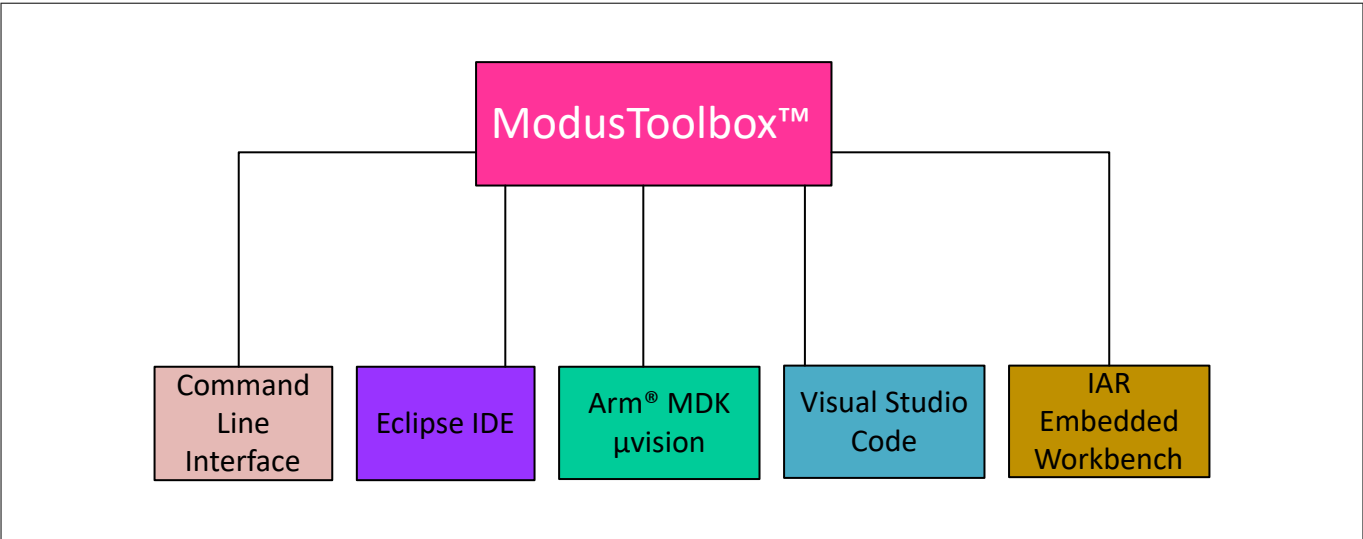
#### 2.2.1 Installing the ModusToolbox™ tools package

Refer to the [ModusToolbox™ tools package installation guide](#) for details.

#### 2.2.2 Choosing an IDE

ModusToolbox™ software, the latest-generation toolset, is supported across Windows, Linux, and macOS platforms. ModusToolbox™ software supports 3rd-party IDEs, including the Eclipse IDE, Visual Studio Code, Arm® MDK (μVision), and IAR Embedded Workbench. The tools package includes an implementation for all the supported IDEs. The tools support all XMC7000 MCUs. The associated BSP and library configurators also work on all three host operating systems.

2 Development ecosystem



**Figure 2** ModusToolbox™ environment

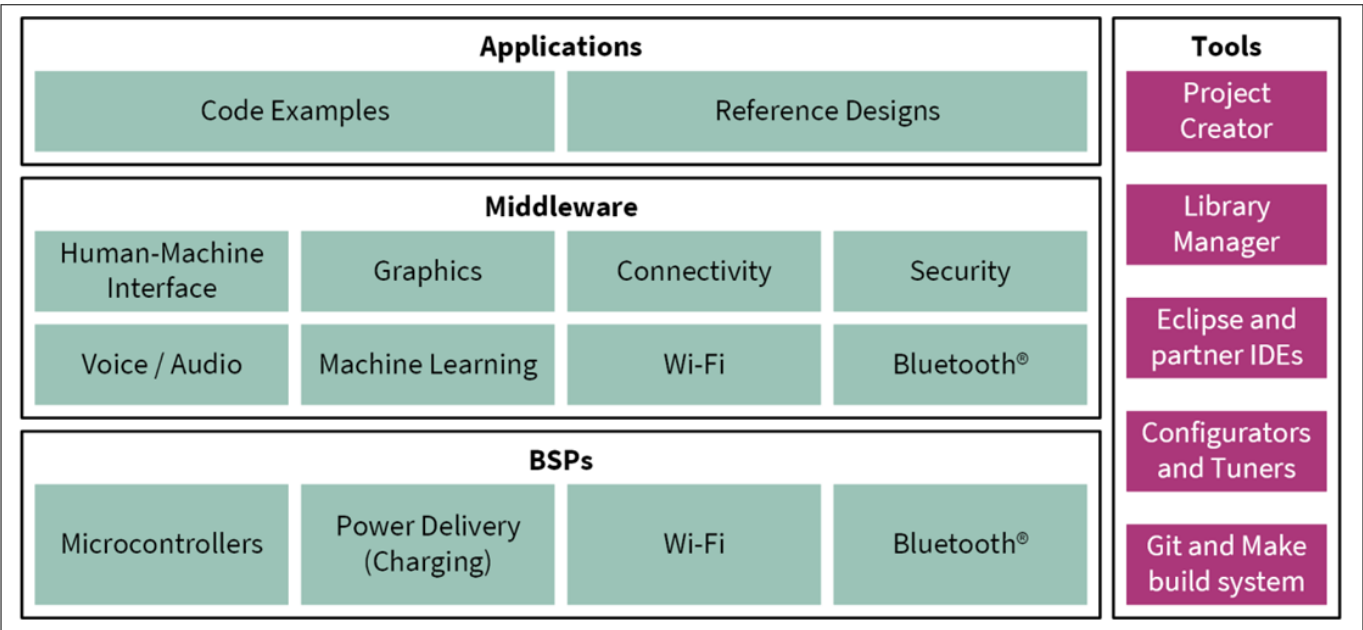
Use ModusToolbox™ to take advantage of the power and extensibility of an Eclipse-based IDE. ModusToolbox™ is supported on Windows, Linux, and macOS. It is recommended to use ModusToolbox™.

**2.2.3** ModusToolbox™ software

ModusToolbox™ software is a set of tools and software that enables an immersive development experience for creating converged MCU and wireless systems, and enables you to integrate our devices into your existing development methodology. These include configuration tools, low-level drivers, libraries, and operating system support, most of which are compatible with Linux-, macOS-, and Windows-hosted environments.

Eclipse IDE for ModusToolbox™ is a multi-platform development environment that supports application configuration and development.

Figure 3 shows a high-level view of the tools/resources included in the ModusToolbox™ software. For a more in-depth overview of the ModusToolbox™ software, see [ModusToolbox™ user guide](#).



**Figure 3** ModusToolbox™ software

## 2 Development ecosystem

The ModusToolbox™ tools package installer includes the design configurators and tools, and the build system infrastructure.

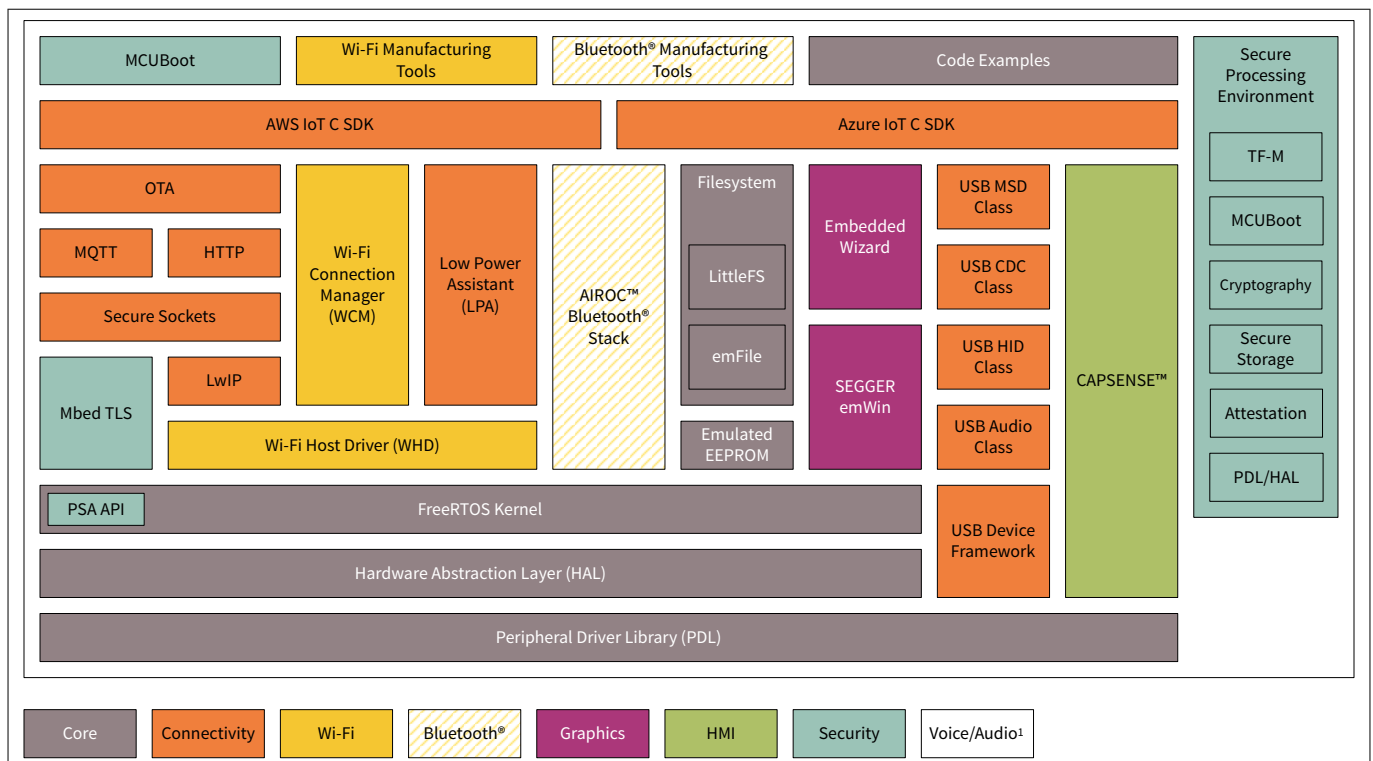
The build system infrastructure includes the new project creation wizard that can be run independent of the Eclipse IDE, the make infrastructure, and other tools.

This means you choose your compiler, IDE, RTOS, and ecosystem without compromising usability or access to our industry-leading CAPSENSE™ (Human-Machine Interface), AIROC™ Wi-Fi and Bluetooth®, security, and various other features.

One part of the ModusToolbox™ ecosystem is run-time software that helps you rapidly develop Wi-Fi and Bluetooth® applications using connectivity combo devices with the XMC™ MCU. See the [ModusToolbox™ run-time software reference guide](#) for details.

Design configurators are the tools that help you create the configurable code for your BSP/middleware. See the [Configurators](#) section for more details.

Figure 4 shows a runtime software diagram to showcase some of the application capabilities of Infineon devices using ModusToolbox™ software.



**Figure 4** ModusToolbox™ run-time software diagram

All the application-level development flows depend on the provided low-level resources. These include:

- Board support packages (BSP) – A BSP is the layer of firmware containing board-specific drivers and other functions. The BSP is a set of libraries that provides APIs to initialize the board and access to board level peripherals. It includes low-level resources such as peripheral driver library (PDL) for XMC7000 MCU and has macros for board peripherals. Custom BSPs can be created to enable support for end-application boards. See [BSP Assistant](#) to create your BSP.
- XMC7000 MCU [peripheral driver library \(PDL\)](#) – The PDL integrates device header files, start-up code, and peripheral drivers into a single package. The PDL supports the XMC7000 MCU family. The drivers abstract the hardware functions into a set of easy-to-use APIs. These are fully documented in the PDL API Reference.



## 2 Development ecosystem

The PDL reduces the need to understand register usage and bit structures, thus easing software development for the extensive set of peripherals in the XMC7000 MCU series. You configure the driver for your application, and then use API calls to initialize and use the peripheral.

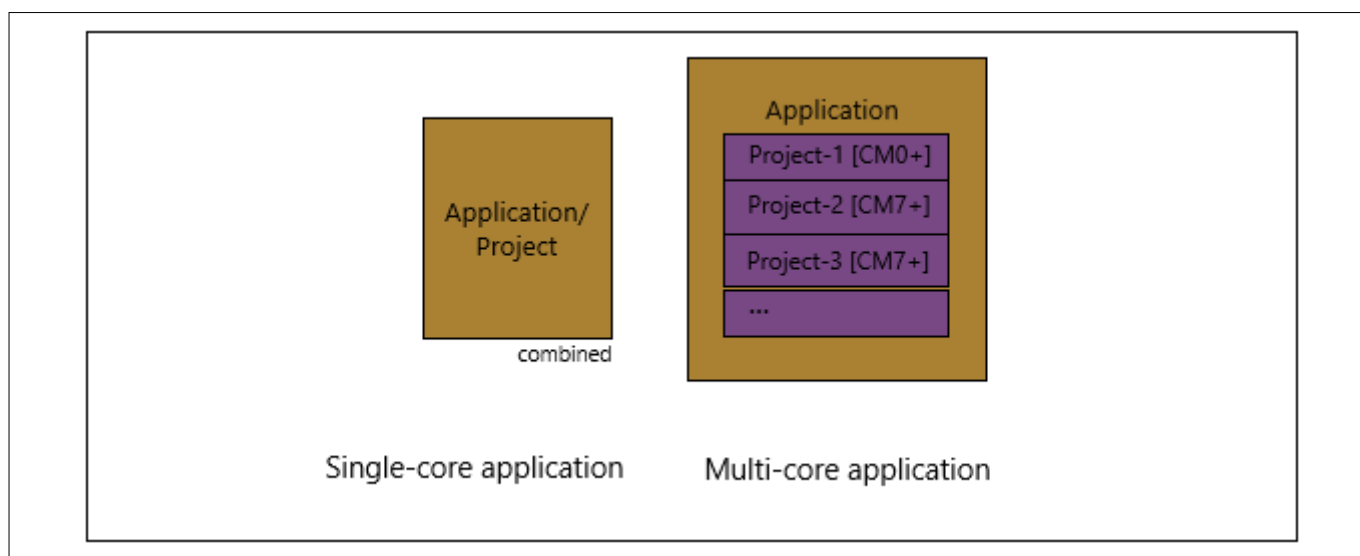
- Middleware (MW) – Extensive middleware libraries that provide specific capabilities to an application. All the middleware is delivered as libraries and via GitHub repositories.

### 2.2.4 ModusToolbox™ applications

With the release of ModusToolbox™ v3.x, multi-core support is introduced, which has altered the folder structure slightly from the previous version of ModusToolbox™.

ModusToolbox™ has two types of applications:

- Single-core application
- Multi-core application



**Figure 5 Application types**

The following shows the new folder structure for an example single-core application:

## 2 Development ecosystem

```
<root>
  ApplicationName
  ->Makefile (MTB_TYPE=COMBINED)
  ->deps
    lib1.mtb (local)
    lib2.mtb (shared)
  ->libs
    lib1 (Infineon Git repo)
  ->bsps
    TARGET_BSP1 (not an Infineon Git repo; completely app-owned)
  ->templates
    TARGET_BSP1
    design.modus
    design.capsense
  ->main.c
  ->helper.h
  ->helper.c
  mtb_shared
    lib2/... (Infineon Git repo)
```

**Figure 6** Folder structure for single-core applications

## 2 Development ecosystem

The following shows the new folder structure for an example multi-core application:

```

<root>
ApplicationName
->Makefile (MTB_TYPE=APPLICATION)
->common.mk
->common_app.mk
->bsps
    TARGET_BSP1 (not an Infineon Git repo; completely app-owned)
->templates
    TARGET_BSP1
        design.modus
        design.capsense
->project1
    Makefile (MTB_TYPE=PROJECT)
    deps
        lib3.mtb (local)
        lib4.mtb (shared)
    libs
        lib3 (Infineon Git repo)
    main.c
    project1_helper.h
    project1_helper.c
->project2
    Makefile (MTB_TYPE=PROJECT)
    deps
        lib5.mtb (local)
        lib6.mtb (shared)
    libs
        lib5 (Infineon Git repo)
    main.c
    project2_helper.h
    project2_helper.c
->project3
    Makefile (MTB_TYPE=PROJECT)
    deps
        lib7.mtb (local)
        lib8.mtb (shared)
    libs
        lib7 (Infineon Git repo)
    main.c
    project3_helper.h
    project3_helper.c
    mtb_shared
        lib4/... (Infineon Git repo)
        lib6/... (Infineon Git repo)
        lib8/... (Infineon Git repo)

```

**Figure 7 Folder Structure for multi-core applications**

The new flow using ModusToolbox™ versions 3.x can support multiple projects in an application. For multi-core applications, there are multiple projects, but only one project per core. The applications have app-owned BSPs, meaning the BSP will be common to all projects inside a multi-core application.

Going further, section 4 of this document describes creating a new single-core application using ModusToolbox™ software.

The project execution starts with executing the code in CM0+ core. In a single core project a pre-compiled program of CM0+ is used which launches CM7\_0 core and goes to sleep. The CM7\_1 core is also put to sleep. In a multi-core project, the application code of all three cores is visible and user can see how code is running in all three cores.

### 2.2.5 XMC7000 software resources

The software for XMC7000 MCUs includes configurators, drivers, libraries, middleware, as well as various utilities, makefiles, and scripts. It also includes relevant drivers, middleware, and examples for use with IoT devices and connectivity solutions. You can use any or all tools in any environment you prefer.

#### 2.2.5.1 Configurators

ModusToolbox™ software provides graphical applications called configurators that make it easier to configure a hardware block. For example, instead of having to search through all the documentation to configure a serial communication block as a UART with a desired configuration, open the appropriate configurator and set the

## 2 Development ecosystem

baud rate, parity, and stop bits. Upon saving the hardware configuration, the tool generates the "C" code to initialize the hardware with the desired configuration.

There are two types of configurators: BSP configurators that configure items that are specific to the MCU hardware and library configurators that configure options for middleware libraries.

Configurators are independent of each other, but they can be used together to provide flexible configuration options. They can be used stand alone, in conjunction with other tools, or within a complete IDE. Configurators are used for:

- Setting options and generating code to configure drivers
- Setting up connections such as pins and clocks for a peripheral
- Setting options and generating code to configure middleware

For XMC7000 MCU applications, the available configurators include:

- **Device Configurator:** Set up the system (platform) functions, pins, and the basic peripherals (e.g., UART, Timer, PWM)
- **QSPI Configurator:** Configure external memory and generate the required code
- **Smart I/O Configurator:** Configure Smart I/O pins

Each of the above configurators creates their own files (e.g., `design.cyqspi` for QSPI). The configurator files (`design.modus` or `design.cyqspi`) are usually provided with the BSP. When an application is created based on a BSP, the files are copied into the application. You can also create custom device configurator files for an application and override the ones provided by the BSP.

### 2.2.5.2 Library management for XMC7000 MCU

The application can have shared/local libraries for the projects. If needed, different projects can use different versions of the same library. The shared libraries are downloaded under the `mtb_shared` directory. The application should use the `deps` folder to add library dependencies. The `deps` folder contains files with the `.mtb` file extension, which is used by ModusToolbox™ to download its git repository. These libraries are direct dependencies of the ModusToolbox™ project.

The Library Manager helps to add/remove/update the libraries of your projects. It also identifies whether particular library has a direct dependency on any other library using the manifest repository available on GitHub, and fetches all its dependencies. These dependency libraries are indirect dependencies of the ModusToolbox™ project. These dependencies can be seen under the `libs` folder. For more information, see the [Library Manager user guide](#) located at `<install_dir> /ModusToolbox/tools_<version>/library-manager/docs/library-manager.pdf`.

### 2.2.5.3 Software development for XMC7000 MCU

The ModusToolbox™ ecosystem provides significant source code and tools to enable software development for XMC7000 MCUs. You use tools to:

- Specify how you want to configure the hardware.
- Generate code for that purpose, which you use in your firmware.
- Include various middleware libraries for additional functionality, like Bluetooth® LE connectivity or FreeRTOS.

This source code makes it easier to develop the firmware for supported devices. It helps you quickly customize and build firmware without the need to understand the register set.

In the ModusToolbox™ environment, you use configurators to configure either the device, or a middleware library, like QSPI functionality.

The XMC7000 MCU Peripheral Driver Library code is delivered as the `mtb-pdl-cat1` library. Middleware is delivered as separate libraries for each feature/function.

## 2 Development ecosystem

In the ModusToolbox™ environment, you use configurators to configure either the device, or a middleware library, like QSPI functionality.

The XMC7000 Peripheral Driver Library code is delivered as the [mtb-pdl-cat1](#) library. Middleware is delivered as separate libraries for each feature/function.

Firmware developers who wish to work at the register level should refer to the driver source code from the PDL. The PDL includes all the device-specific header files and startup code you need for your project. It also serves as a reference for each driver. Because the PDL is provided as source code, you can see how it accesses the hardware at the register level.

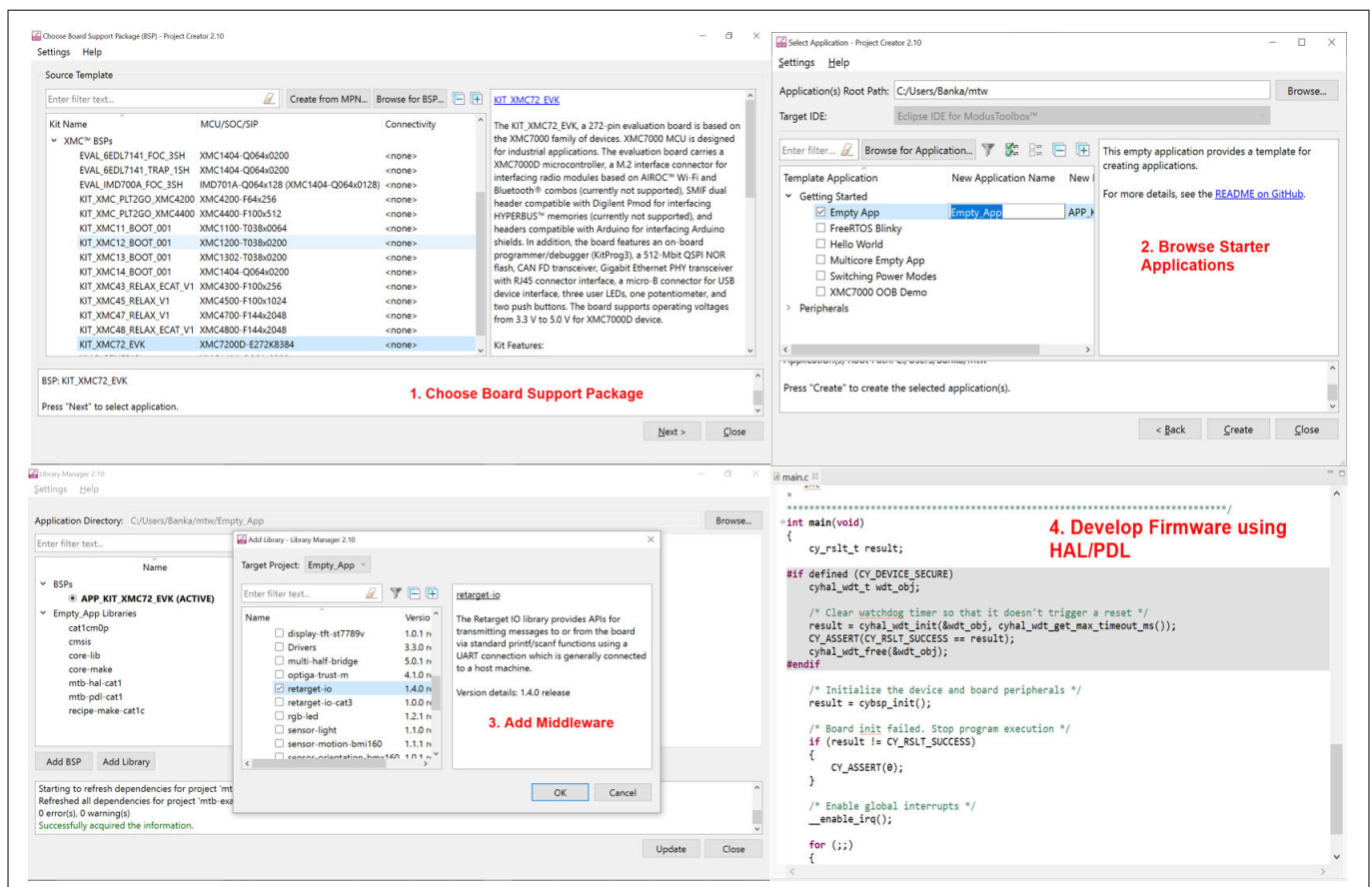
Some devices do not support particular peripherals. The PDL is a superset of all the drivers for any supported device. This superset design means:

- All API elements needed to initialize, configure, and use a peripheral are available.
- The PDL is useful across various XMC7000 MCUs, regardless of available peripherals.
- The PDL includes error checking to ensure that the targeted peripheral is present on the selected device.

This enables the code to maintain compatibility across products of the XMC7000 MCU family, as long as the peripherals are available. A device header file specifies the peripherals that are available for a device. If you write code that attempts to use an unsupported peripheral, you will get an error at compile time. Before writing code to use a peripheral, consult the datasheet for the particular device to confirm support for that peripheral.

As the following figure shows, with the ModusToolbox™ software, you can:

1. Choose a BSP (Project Creator).
2. Create a new application based on a list of starter applications, filtered by the BSPs that each application supports (Project Creator).
3. Add BSP or middleware libraries (Library Manager).
4. Develop your application firmware using PDL for XMC7000 MCU (IDE of choice or command line).



**Figure 8** Eclipse IDE for ModusToolbox™ resources and middleware

---

## 2 Development ecosystem

### 2.3 Support for other IDEs

You can develop firmware for XMC7000 MCUs using your preferred IDE such as Eclipse IDE, [IAR Embedded Workbench](#), [Keil µVision 5](#), or [Visual Studio Code](#).

ModusToolbox™ Configurators are stand-alone tools that can be used to set up and configure XMC7000 MCU resources and other middleware components without using the Eclipse IDE. The Device Configurator and middleware configurators use the `design.x` files within the application workspace. You can then point to the generated source code and continue developing firmware in your IDE.

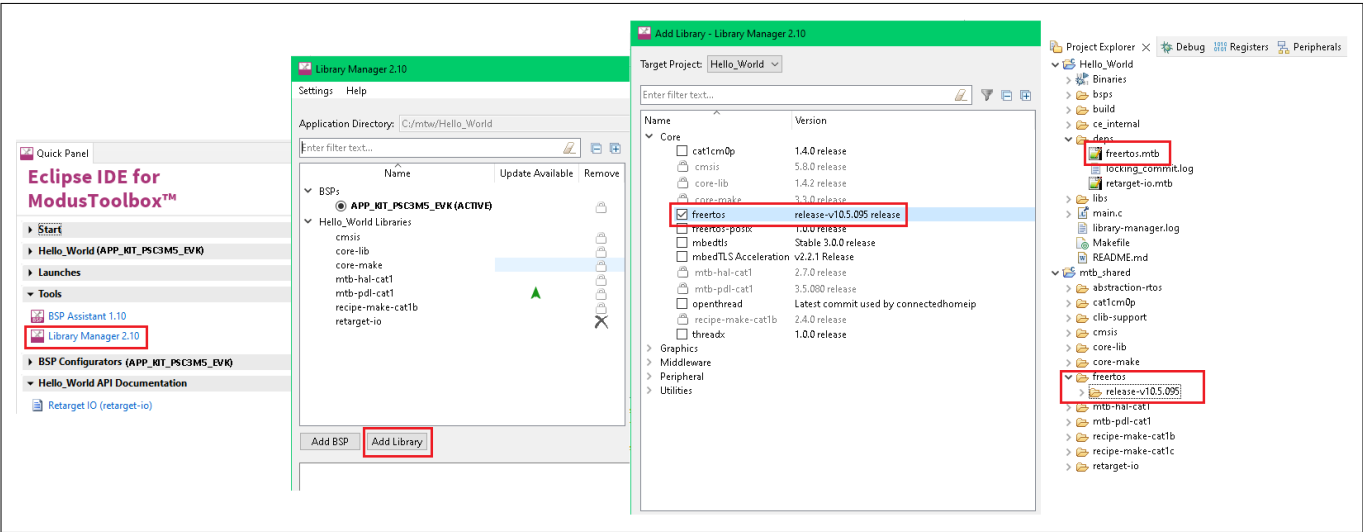
If there is a change in the device configuration, edit the `design.x` files using the configurators and regenerate the code. It is recommended that you generate resource configurations using the configuration tools provided with ModusToolbox™ software.

## 2 Development ecosystem

### 2.4 FreeRTOS support with ModusToolbox™

Adding native FreeRTOS support to a ModusToolbox™ application project is like adding any middleware library. You can include the FreeRTOS middleware in your application by using the Library Manager. If using the Eclipse IDE for ModusToolbox™, select the application project and click the **Library Manager** link in the **Quick Panel**. Click **Add Library** and select **freertos** from the **Core** dialog, as [Figure 9](#) shows.

The .mtb file pointing to the FreeRTOS middleware is added to the application project's deps directory. The middleware content is also downloaded and placed inside the corresponding folder called **freertos**. The default location is in the shared asset repo named mtb\_shared. To continue working with FreeRTOS, follow the steps in the Quick Start section of [FreeRTOS documentation](#).



**Figure 9** Import FreeRTOS middleware in ModusToolbox™ application

### 2.5 Programming/debugging using Eclipse IDE

All XMC7000 Kits have a KitProg3 onboard programmer/debugger. It supports Cortex® Microcontroller Software Interface Standard - Debug Access Port (CMSIS-DAP). See the KitProg3 user guide for details.

The Eclipse IDE requires KitProg3 and uses the OpenOCD protocol for debugging XMC7000 MCU applications. It also supports GDB debugging using industry standard probes like the Segger J-Link.

ModusToolbox™ includes the **fw-loader** command-line tool to update and switch the KitProg firmware from KitProg2 to KitProg3. Refer to the “XMC7000 Programming/Debugging - KitProg Firmware Loader” section in the Eclipse IDE for ModusToolbox™ user guide for more details.

For more information on debugging firmware on XMC7000 devices with ModusToolbox™, refer to the Program and Debug section in the Eclipse IDE for ModusToolbox™ user guide.

### 2.6 XMC7000 MCU development kits

**Table 2** Development kits

Product line	Development kits
Performance	<a href="#">XMC7200 Evaluation Kit (KIT_XMC72_EVK)</a>

For the complete list of kits for the XMC7000 MCU along with the shield modules, see the [Microcontroller \(MCUs\) kits](#) page.



### 3 Device features

## 3 Device features

XMC7000 MCUs have extensive features as shown in [Figure 10](#). The following is a list of major features. For more information, see the device [datasheet](#), the [technical reference manual \(TRM\)](#), and the section on [References](#).

- **CPU Subsystem**
  - One or two 250-MHz/350-MHz Arm™ Cortex™-M7 and 100-MHz Arm™ Cortex™-M0+
  - Inter-processor communication supported in hardware
  - Three DMA controllers
- **Integrated memories**
  - Up to 8384 KB of code flash with an additional 256 KB work flash
  - Up to 1024 KB of SRAM selectable retention granularity
- **Cryptography engine**
  - Supports Enhanced Secure Hardware Extension (eSHE) and Hardware Security Module (HSM)
  - Secure boot and authentication
  - AES: 128-bit blocks, 128-/192-/256-bit keys
  - 3DES: 64-bit blocks, 64-bit key
  - Vector unit supporting asymmetric key cryptography such as Rivest-Shamir-Adleman (RSA) and Elliptic Curve (ECC)
  - SHA-1/2/3: SHA-512, SHA-256, SHA-160 with variable length input data
  - CRC: supports CCITT CRC16 and IEEE-802.3 CRC32
  - True random number generator (TRNG) and pseudo random number generator (PRNG)
  - Galois/Counter Mode (GCM)
- **Safety for application**
  - Memory Protection Unit (MPU)
  - Shared Memory Protection Unit (SMPU)
  - Peripheral Protection Unit (PPU)
  - Watchdog Timer (WDT)
  - Multi-counter Watchdog Timer (MCWDT)
  - Low-voltage Detector (LVD)
  - Brown-out Detection (BOD)
  - Overvoltage Detection (OVD)
  - Clock Supervisor (CSV)
  - Hardware error correction (SECDED ECC) on all safety-critical memories (SRAM, flash, TCM)
- **Low-power 2.7-V to 5.5-V operation**
  - Low-power Active, Sleep, Low-power Sleep, DeepSleep, and Hibernate modes for fine-grained power management
  - Configurable options for robust BOD
- **Wakeup**
  - Up to two pins to wake from Hibernate mode
  - Up to 220 GPIO pins to wake from Sleep modes
  - Event Generator, SCB, Watchdog Timer, RTC alarms to wake from DeepSleep modes
- **Clocks**
  - Internal Main Oscillator (IMO)
  - Internal Low-Speed Oscillator (ILO)

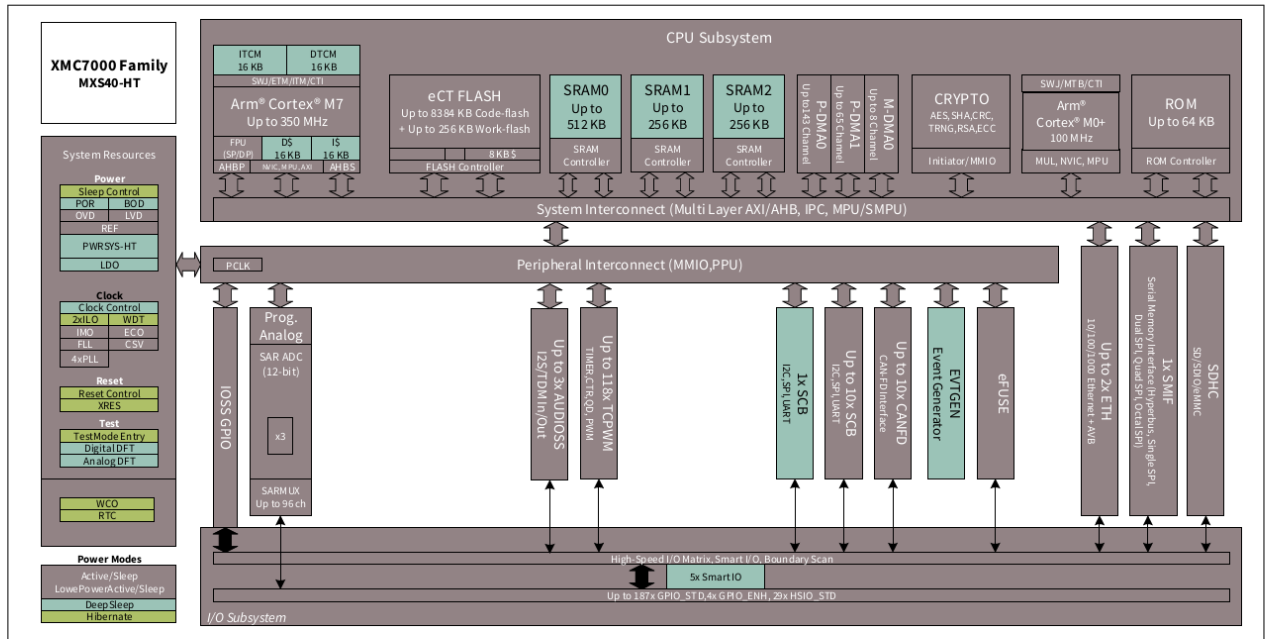


## 3 Device features

---

- External Crystal Oscillator (ECO)
- Watch Crystal Oscillator (WCO)
- Phase-Locked Loop (PLL)
- Frequency-Locked Loop (FLL)
- **Communication interfaces**
  - Up to 10 CAN FD channels
  - Up to 11 runtime-reconfigurable SCB (serial communication block) channels, each configurable as I2C, SPI, or UART
  - Up to two 10/100/1000 Mbps Ethernet MAC interfaces conforming to IEEE-802.3az
- **External memory interface**
  - One SPI (Single, Dual, Quad, or Octal) or HYPERBUS™ interface
  - On-the-fly encryption and decryption
  - Execute-In-Place (XIP) from external memory
- **SDHC interface**
  - One Secure Digital High Capacity (SDHC) interface supporting embedded MultiMediaCard (eMMC), Secure
  - Digital (SD), or SDIO (Secure Digital Input Output)
  - Data rates up to SD High-Speed 50 MHz, or eMMC 52 MHz DDR
- **Audio interface**
  - Three Inter-IC Sound (I2S) Interfaces (based on the NXP I2S bus specification) for connecting digital audio devices
  - I2S, left justified, or Time Division Multiplexed (TDM) audio formats
  - Independent transmit or receive operation, each in master or slave mode
- **Timers**
  - Up to 102 blocks of 16-bit and 16 blocks of 32-bit Timer/Counter Pulse-Width Modulator (TCPWM)
  - Up to 16 Event Generation (EVTGEN) timers supporting cyclic wakeup from DeepSleep
- **Real time clock (RTC)**
  - Year/Month/Date, Day-of-week, Hour:Minute:Second fields
  - 12- and 24-hour formats
  - Automatic leap-year correction
- **I/O**
  - Clock supervisor (CSV)
  - Three I/O types: GPIO\_STD/GPIO\_ENH/HSIO\_STD
- **Smart I/O**
  - Up to five smart I/O blocks, which can perform Boolean operations on signals going to and from I/Os
  - Up to 36 I/Os (GPIO\_STD) supported
- **I/O subsystem**
  - Up to 220 GPIOs with programmable drive modes, drive strength, slew rates
  - Two ports with smart I/O that can implement Boolean operations

## 3 Device features



**Figure 10 XMC7000 MCU block diagram**

### • Programmable analog

- Three SAR A/D converters with up to 99 external channels (96 I/Os + 3 I/Os for motor control)
- Each ADC supports 12-bit resolution and sampling rates of up to 1 Msps
- Each ADC also supports six internal analog inputs
- Each ADC supports addressing of external multiplexers
- Each ADC has a sequencer supporting autonomous scanning of configured channels
- Synchronized sampling of all ADCs for motor-sense applications

## 4 Getting started with XMC7000 MCU design

### 4 Getting started with XMC7000 MCU design

This section provides the following:

- Demonstrate how to build a simple XMC7000 MCU-based design and program it on to the development kit
- Makes it easy to learn XMC7000 MCU design techniques and how to use the ModusToolbox™ software with different IDEs.

**Note:** You can use any supported IDE, but this section uses the Eclipse IDE as an example.

#### 4.1 Prerequisites

Before you get started, make sure that you have the appropriate development kit for your XMC7000 MCU product line and have installed the required software. You also need internet to access the GitHub repositories during project creation.

##### 4.1.1 Hardware

The example design shown below is developed for the [XMC7200 Evaluation Kit](#). However, you can build the application for other development kits. See the [Using these instructions](#) section.

##### 4.1.2 Software

[ModusToolbox™ software](#) 3.2 or above.

After installing the software, see the [ModusToolbox™ tools package user guide](#) to get an overview of the software.

#### 4.2 Using these instructions

These instructions are grouped into several sections. Each section is dedicated to a phase of the application development workflow. The major sections are:

1. [Create a new application](#)
2. [View and modify the design](#)
3. [Write firmware](#)
4. [Build the application](#)
5. [Program the device](#)
6. [Test your design](#)

This design is developed for the [XMC7200 Evaluation Kit](#). You can use other supported kits to test this example by selecting the appropriate kit while creating the application.

#### 4.3 About the design

This design uses the CM7 CPU of the XMC7000 MCU to execute two tasks: UART communication and LED control.

After device reset, the CM7 CPU uses the UART to print a “Hello World” message to the serial port stream, and starts blinking the user LED on the kit. When you press the Enter key on the serial console, the blinking is paused or resumed.

## 4 Getting started with XMC7000 MCU design

### 4.4 Create a new application

This section takes you on a step-by-step guided tour of the new application process. It uses the starter application and manually adds the functionality from the **Hello World** starter application. The Eclipse IDE for ModusToolbox™ is used in the instructions, but you can use any IDE or the command line if you prefer.

If you are familiar with developing projects with ModusToolbox™ software, you can use the **Hello World** starter application directly. It is a complete design, with all the firmware written for the supported kits. You can walk through the instructions and observe how the steps are implemented in the code example.

If you start from scratch and follow all the instructions in this application note, you can use the **Hello World** code example as a reference while following the instructions.

Launch the Dashboard 3.2 application to get started.

**Note:** *Dashboard 3.2 application needs access to the internet to successfully clone the starter application onto your machine.*

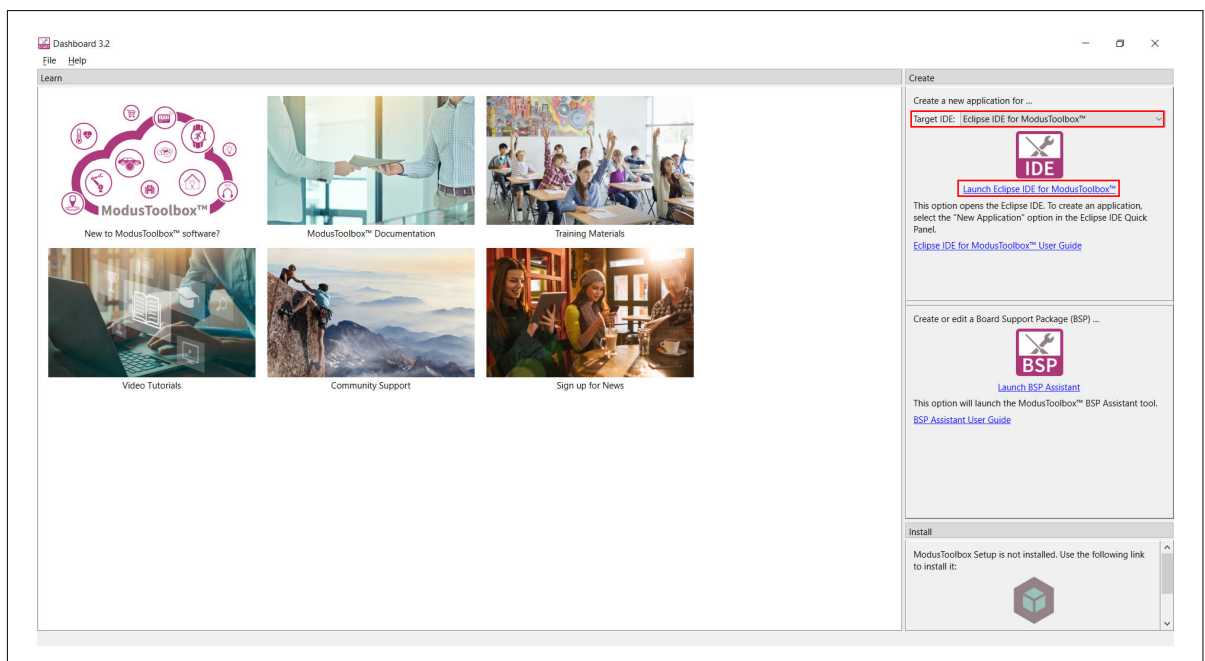
The Dashboard 3.2 application helps you get started using the various tools with easy access to documentation and training material, a simple path for creating applications and creating and editing BSPs.

**1.** Open the Dashboard 3.2 application.

To open the Dashboard 3.2 application, do one of these:

- **Windows:** Navigate to [ModusToolbox installation path]/tools\_3.2/dashboard/dashboard.exe Or you can also select the "dashboard" item from the Windows Start menu.
- **Linux:** [ModusToolbox installation path]/tools\_3.2/dashboard and run the executable
- **macOS:** Run the "dashboard" app

**2.** On the Dashboard 3.2 window, in the right pane, in the **Target IDE** drop-down list, select **Eclipse IDE for ModusToolbox™**, and click **Launch Eclipse IDE for ModusToolbox™**.



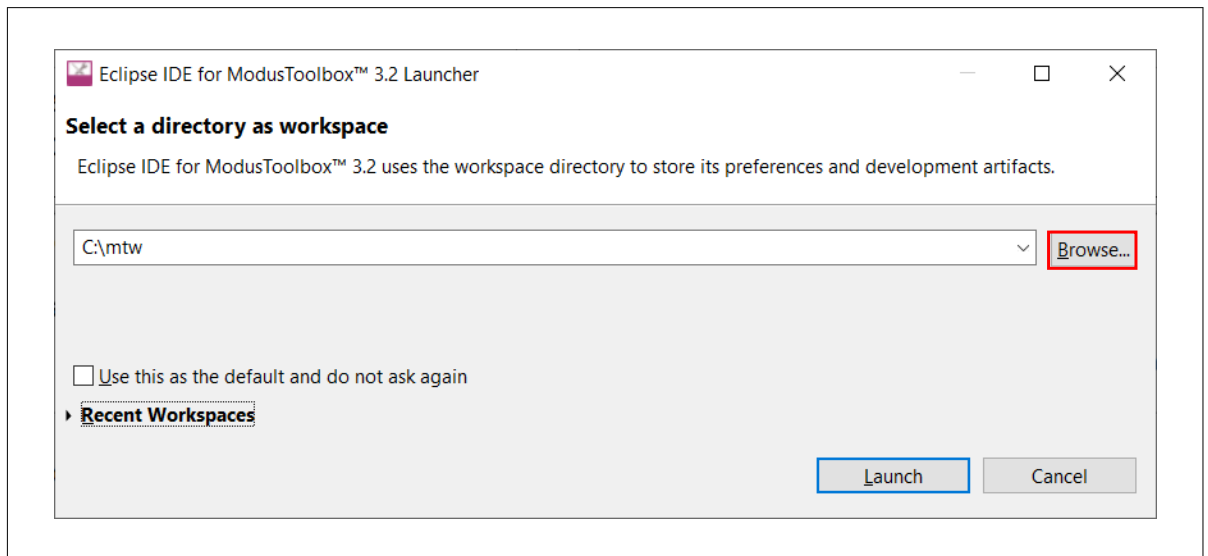
**Figure 11** Dashboard 3.2 application

**3.** Select a new workspace.

At launch, Eclipse IDE for ModusToolbox™ displays a dialog to choose a directory for use as the workspace directory. The workspace directory is used to store workspace preferences and development artifacts. You can choose an existing empty directory by clicking the **Browse** button, as shown in the

## 4 Getting started with XMC7000 MCU design

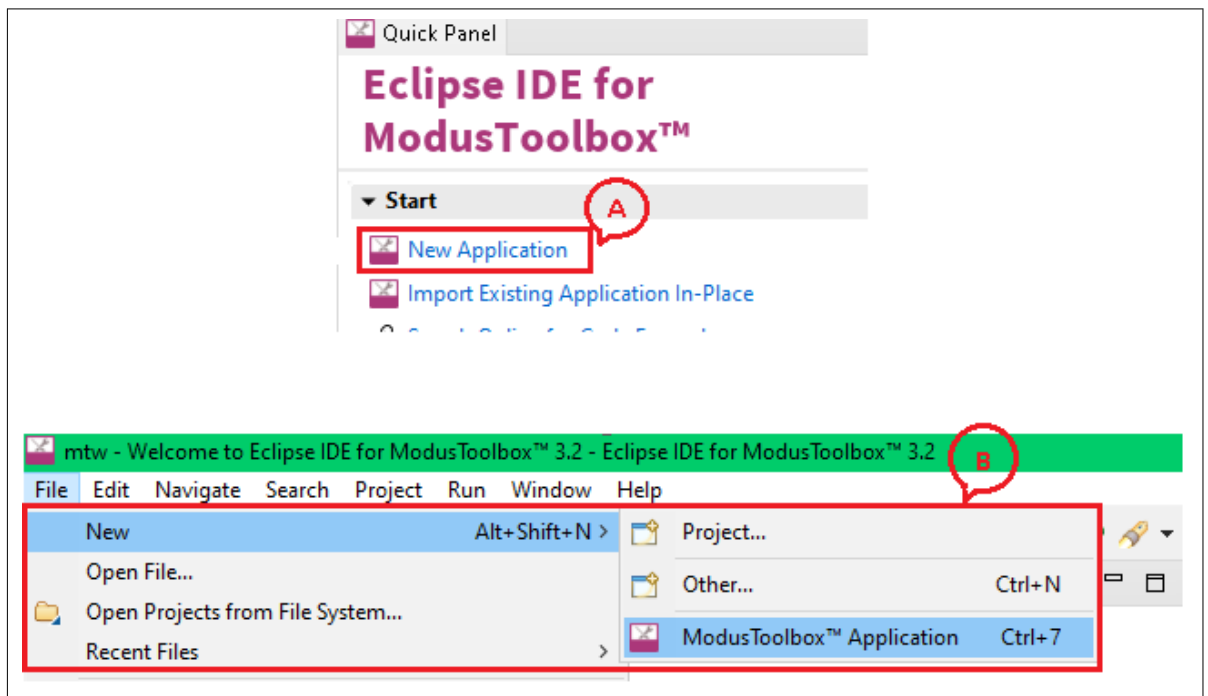
following figure. Alternatively, you can type in a directory name to be used as the workspace directory along with the complete path, and the IDE will create the directory for you.



**Figure 12** Select a directory as the workspace

### 4. Create a new ModusToolbox™ application.

- Click **New Application** in the Start group of the Quick Panel.
- Alternatively, you can choose **File > New > ModusToolbox Application**, as Figure 13 shows. The Eclipse IDE for ModusToolbox™ Application window appears.



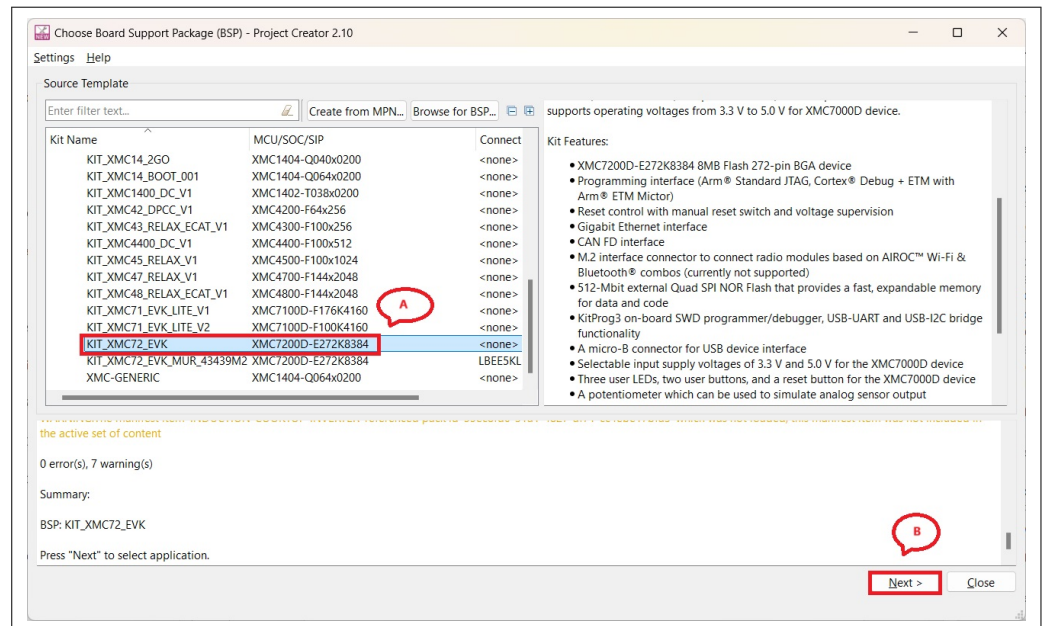
**Figure 13** Create a New ModusToolbox™ Application

### 5. Select a target XMC7200 evaluation kit.

ModusToolbox™ speeds up the development process by providing BSPs that set various workspace/project options for the specified development kit in the new application dialog.

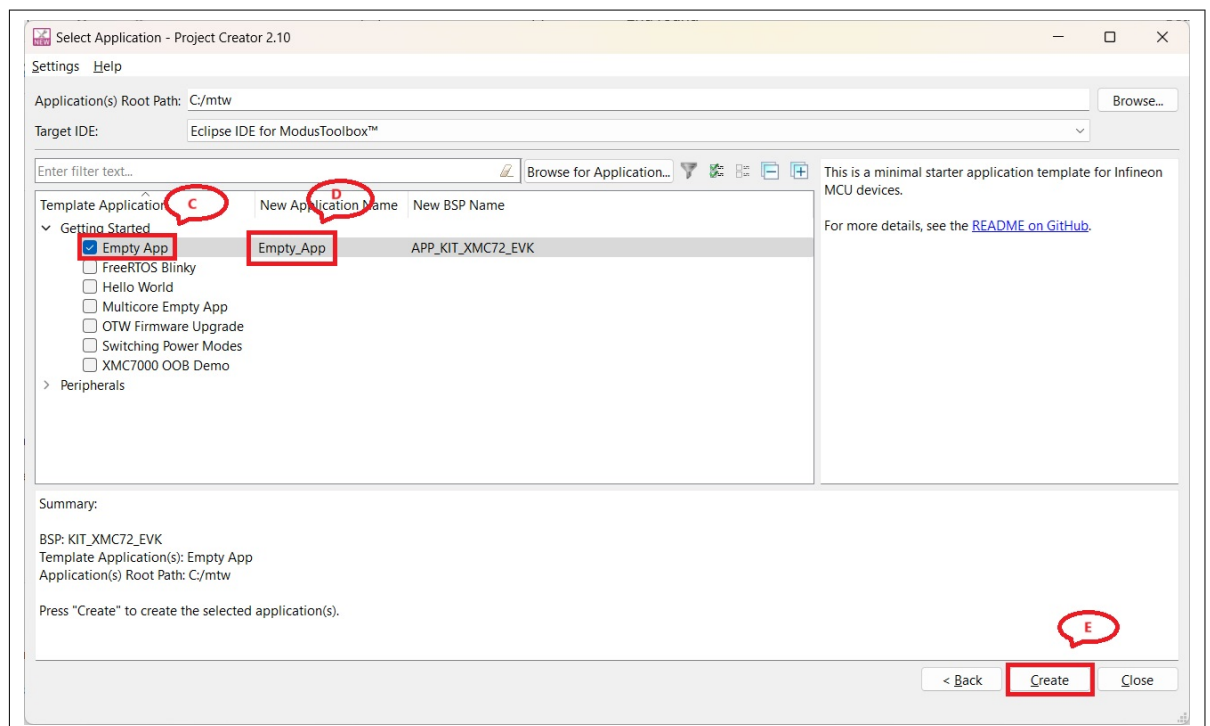
- In the **Choose Board Support Package (BSP)** dialog, choose the **Kit Name** that you have. The steps that follow use **KIT\_XMC72\_EVK**. See Figure 14 for help with this step.
- Click **Next**.

## 4 Getting started with XMC7000 MCU design



**Figure 14 Choose target hardware**

- In the **Select Application** dialog, select **Empty App** starter application, as shown in the following figure.
- In the **Name** field, type in a name for the application, such as **Hello\_World**. You can choose to leave the default name if you prefer.
- Click **Create** to create the application, as shown in the following figure, wait for the Project Creator to automatically close once the project is successfully created.



**Figure 15 Choose single core starter application**

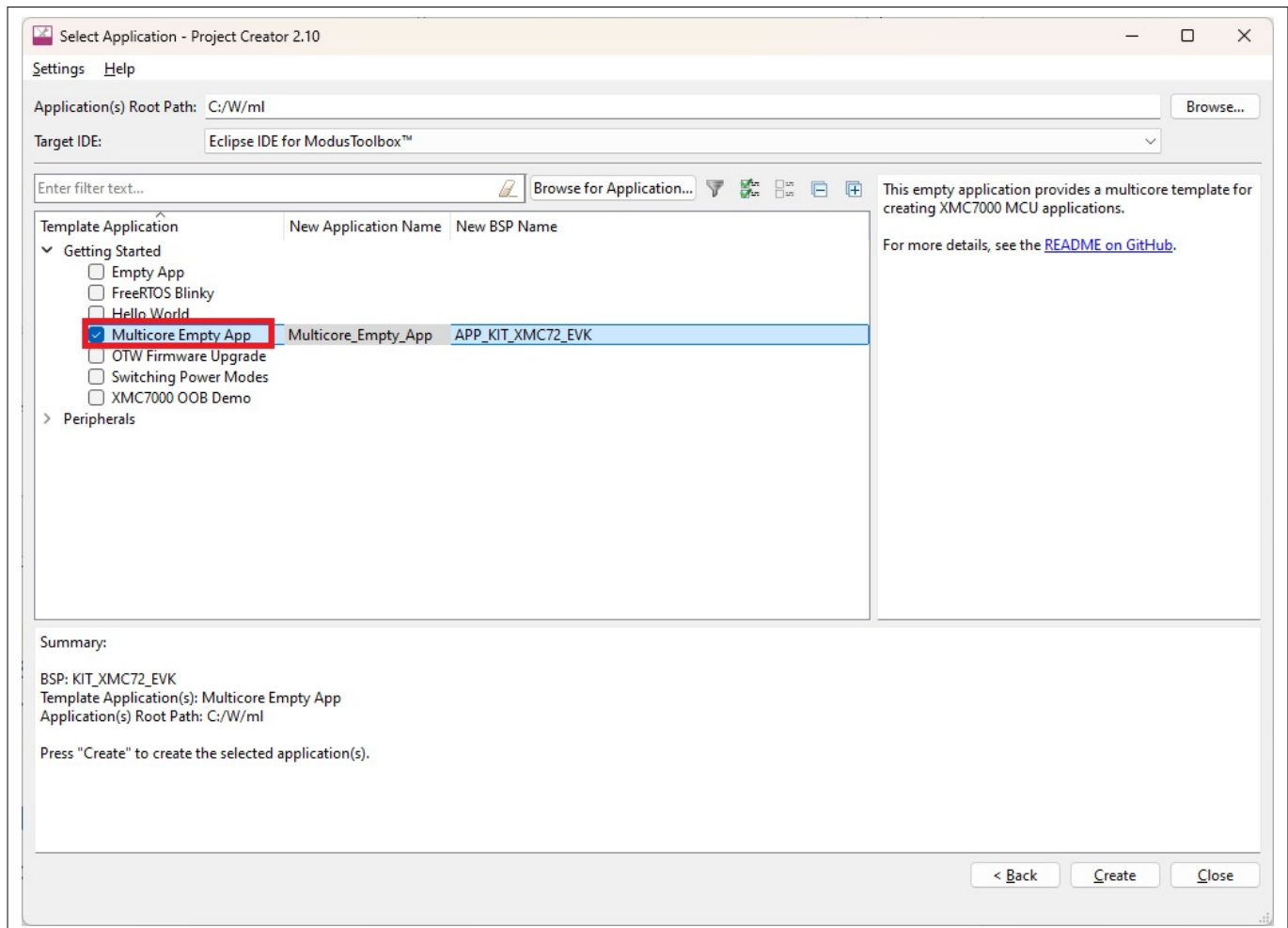
You have successfully created a new ModusToolbox™ application for a XMC7000 MCU.

The BSP uses XMC7200D-E272K8384 as the default device that is mounted on the [XMC7200 Evaluation Kit](#).

## 4 Getting started with XMC7000 MCU design

If you are using custom hardware based on XMC7000 MCU, or a different XMC7000 MCU part number, see the "Creating your Own BSP" section in the [ModusToolbox™ user guide](#).

**Note:** To create a multi core project, select **Multicore Empty App** from **Select Application** dialog as shown in below [Figure 16](#). Refer to [ModusToolbox™ applications](#) for more details.



**Figure 16** Choose multi core starter application

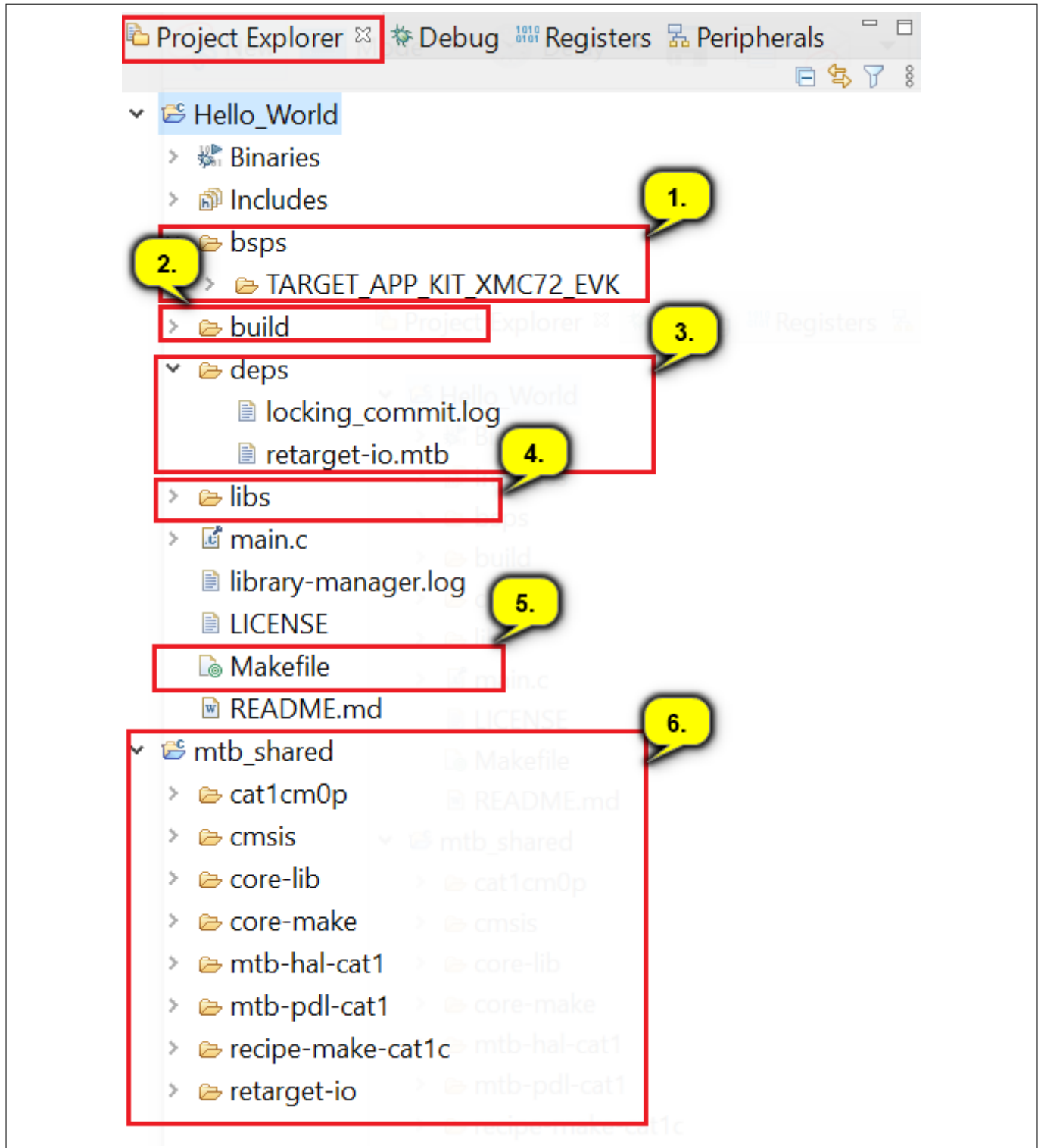
### 4.5 View and modify the design

[Figure 17](#) shows the ModusToolbox™ project explorer interface displaying the structure of the application project.

XMC7000 MCU consists of three cores: one CM0+ core and two CM7 cores. This application note shows the firmware development using the CM7 core with ModusToolbox™ software.



#### 4 Getting started with XMC7000 MCU design



**Figure 17** Project Explorer view

A project folder consists of various subfolders – each denoting a specific aspect of the project.

1. The files provided by the BSP are in the `bsps` folder and are listed under `TARGET_<bsp name>` subfolders. All the input files for the device and peripheral configurators are in the `config` folder inside the BSP. The `GeneratedSource` folder in the BSP contains the files that are generated by the configurators and are prefixed with `cycfg_`. These files contain the design configuration as defined by the BSP. From ModusToolbox™ 3.x or later, you can directly customize configurator files of BSP for your application



## 4 Getting started with XMC7000 MCU design

rather than overriding the default design configurator files with custom design configurator files since BSPs are completely owned by the application.

The BSP folder also contains the linker scripts and the start-up code for the XMC7000 MCU used on the board.

2. The build folder contains all the artifacts resulting from a build of the project. The output files are organized by target BSPs.
3. The `deps` folder contains `.mtb` files, which provide the locations from which ModusToolbox™ pulls the libraries that are directly referenced by the application. These files typically each contain the GitHub location of a library. The `.mtb` files also contain a git Commit Hash or Tag that tells which version of the library is to be fetched and a path as to where the library should be stored locally.

For example, here, `retarget-io.mtb` points to `mtb://retarget-io#latest-v1.X$$$ASSET_REPO$$$retarget-io/latest-v1.x`. The variable `$$$ASSET_REPO$$$` points to the root of the shared location which defaults to `mtb_shared`. If the library must be local to the application instead of shared, use `$$$LOCAL$$$` instead of `$$$ASSET_REPO$$$`.

4. The `libs` folder also contains `.mtb` files. In this case, they point to libraries that are included indirectly as a dependency of a BSP or another library. For each indirect dependency, the Library Manager places a `.mtb` file in this folder. These files have been populated based on the targets available in `deps` folder. For example, using BSP `KIT_XMC72_EVK` populates `libs` folder with the following `.mtb` files: [cmsis.mtb](#), [core-lib.mtb](#), [core-make.mtb](#), [mtb-hal-cat1.mtb](#), [mtb-pdl-cat1.mtb](#), [cat1cm0p.mtb](#), [recipe-make-cat1a.mtb](#).

The `libs` folder contains the file `mtb.mk`, which stores the relative paths of all the libraries required by the application. The build system uses this file to find all the libraries required by the application.

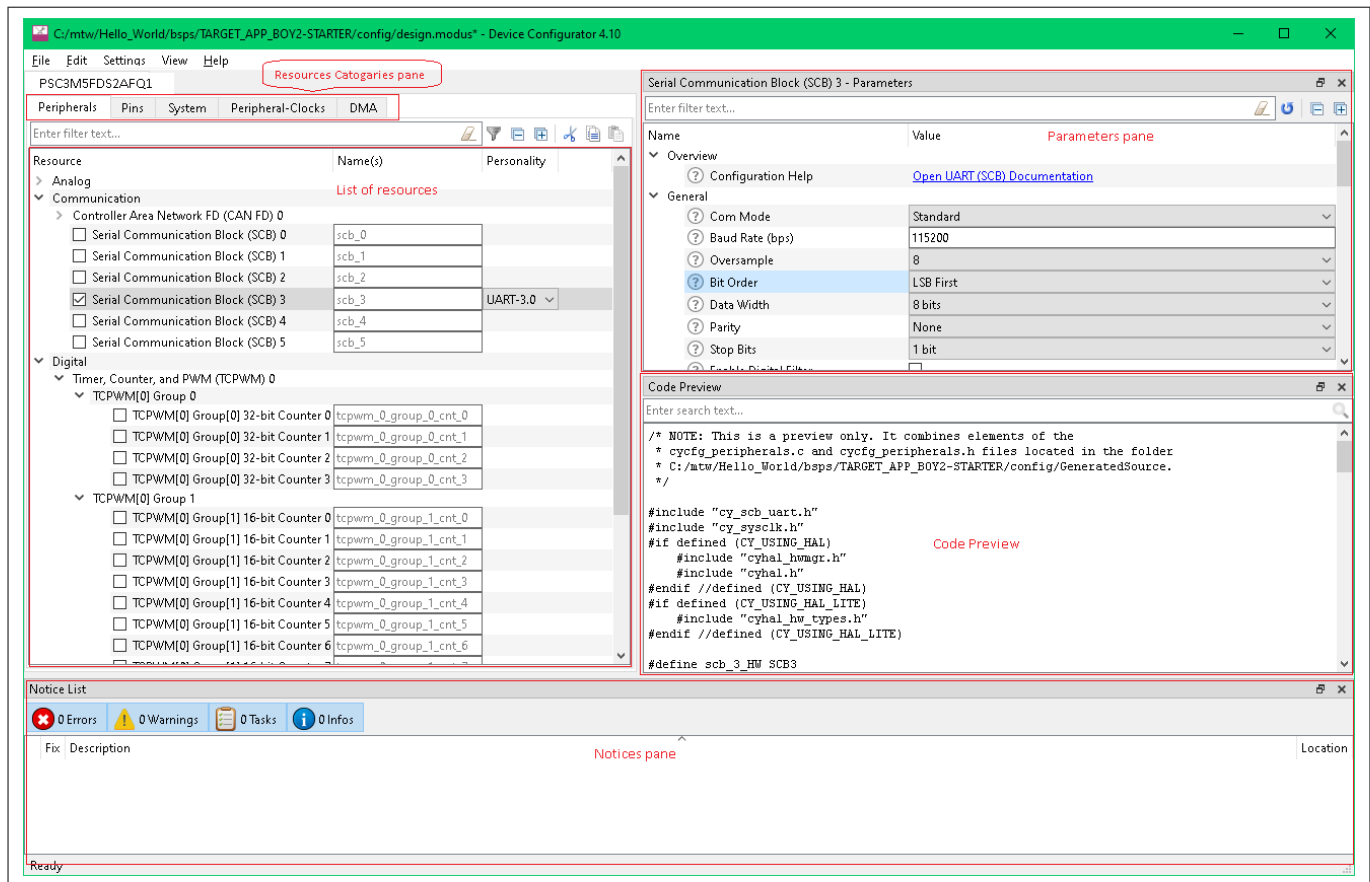
Everything in the `libs` folder is generated by the Library Manager so you should not manually edit anything in that folder.

5. An application contains a Makefile which is at the application's root folder. This file contains the set of directives that the make tool uses to compile and link the application project. There can be more than one project in an application. In that case there is a Makefile at the application level and one inside each project.
6. By default, when creating a new application or adding a library to an existing application and specifying it as shared, all libraries are placed in an `mtb_shared` directory adjacent to the application directories. The `mtb_shared` folder is shared between different applications within a workspace. Different applications may use different versions of shared libraries if necessary.

### 4.5.1 Open the Device Configurator

BSP configurator files are in the `bsps/TARGET_<BSP-name>/config` folder. For example, click **<Application-name>** from **Project Explorer** then click **Device Configurator** link in the **Quick Panel** to open the file `design.modus` in the **Device Configurator** as shown in the following figure. You can also open other configuration files in their respective configurators or click the corresponding links in the **Quick Panel**.

## 4 Getting started with XMC7000 MCU design



**Figure 18** Device Configurator

The **Device Configurator** provides a set of **Resources Categories** tabs. Here you can choose between different resources available in the device such as peripherals, pins, and clocks from the **List of Resources**.

You can choose how a resource behaves by choosing a **Personality** for the resource. For example, a **serial communication block (SCB)** resource can have **EZIC**, **I2C**, **SPI**, or **UART** personalities. The **Alias** is your name for the resource, which is used in firmware development. One or more aliases can be specified by using a comma to separate them (with no spaces).

The **Parameters** pane is where you enter the configuration parameters for each enabled resource and the selected personality. The **Code Preview** pane shows the configuration code generated per the configuration parameters selected. This code is populated in the `cycfg_` files in the `GeneratedSource` folder. The Parameters pane and Code Preview pane may be displayed as tabs instead of separate windows but the contents will be the same.

Any errors, warnings, and information messages arising out of the configuration are displayed in the **Notices** pane.

The application project contains source files that help you create an application for the CM7 core (for example, `main.c`). This c file is compiled and linked with the CM7 image as part of the normal build process.

At this point in the development process, the required middleware is ready to be added to the design. The only middleware required for the Hello World application is the [retarget-io](#) library.

### 4.5.2 Add retarget-io middleware

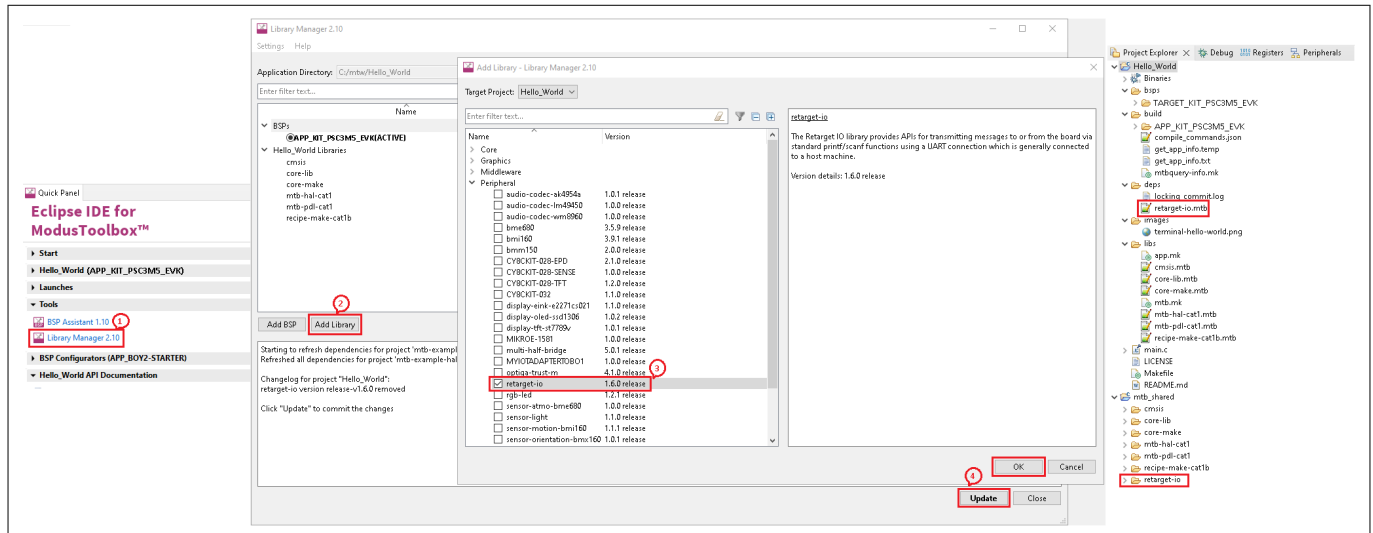
In this section, you will add the [retarget-io](#) middleware to redirect standard input and output streams to the UART configured by the BSP. The initialization of the middleware will be done in `main.c` file.

1. In the **Quick Panel**, click the **Library Manager** link.

## 4 Getting started with XMC7000 MCU design

2. In the subsequent dialog, click **Add Libraries**.
3. Under **Peripherals**, select and enable **retarget-io**.
4. Click **OK** and then **Update**.

The files necessary to use the **retarget-io** middleware are added in the `mtb_shared > retarget_io` folder, and the `.mtb` file is added to the `deps` folder, as shown in the following figure.



**Figure 19** Add the **retarget-io** middleware

### 4.5.3 Configuration of UART, timer peripherals, pins, and system clocks

The configuration of the debug UART peripheral, timer peripheral, pins, and system clocks can be done directly in the code using the function APIs provided by the BSP and HAL. Therefore, it is not necessary to configure them with the Device Configurator. See [Write firmware](#) section for more details.

## 4.6 Write firmware

At this point in the development process, you have created an application with the assistance of an application template and modified it to add the **retarget-io** middleware. In this section, you will write the firmware that implements the design functionality.

If you are working from scratch using the [Empty XMC7000](#) starter application, you can copy the respective source code to the `main.c` file of the application project from the code snippet provided in this section. If you are using the [Hello World](#) code example, all the required files are already in the application.

### Firmware flow

Examine the code in the `main.c` file of the application. [Figure 20](#) shows the firmware flowchart.

After reset, resource initialization for this example is performed by the CM7 CPU. It configures the system clocks, pins, clock to peripheral connections, and other platform resources.

After reset, the clocks and system resources are initialized by the BSP initialization function. The **retarget-io** middleware is configured to use the debug UART, and the user LED is initialized. The debug UART prints a “Hello World!” message on the terminal emulator – the onboard KitProg3 acts as the USB-to-UART bridge to create the virtual COM port. A timer object is configured to generate an interrupt every 1000 milliseconds. At each timer interrupt, the CM7 CPU toggles the LED state on the kit.

The firmware is designed to accept the 'Enter' key as an input, and on every press of the 'Enter' key, the firmware starts or stops the blinking of the LED.

Note that this application code uses BSP/HAL/middleware functions to execute the intended functionality.

---

### 4 Getting started with XMC7000 MCU design

`cybsp_init()` – This BSP function sets up the HAL hardware manager and initializes all the system resources of the device, including but not limited to the system clocks and power regulators.

`cy_retarget_io_init()` – This function from the [retarget-io](#) middleware uses the aliases set up in the BSP for the debug UART pins to configure the debug UART with a standard baud rate of 115200 and also redirects the input/output stream to the debug UART.

**Note:** *You can open the Device Configurator to view the aliases that are set up in the BSP.*

`cyhal_gpio_init()` – This function from the GPIO HAL initializes the physical pin to drive the LED. The LED used is derived from the alias for the pin set up in the BSP.

`timer_init()` – This function wraps a set of timer HAL function calls to instantiate and configure a hardware timer. It also sets up a callback for the timer interrupt.

Copy the following code snippet to the `main.c` file of your application project.

## 4 Getting started with XMC7000 MCU design

### Code listing 1: main.c file

```
#include "cyhal.h"
#include "cybsp.h"
#include "cy_retarget_io.h"

/*****
 * Macros
 *****/

/* LED blink timer clock value in Hz */
#define LED_BLINK_TIMER_CLOCK_HZ      (10000)

/* LED blink timer period value */
#define LED_BLINK_TIMER_PERIOD        (9999)

/*****
 * Function Prototypes
 *****/

void timer_init(void);
static void isr_timer(void *callback_arg, cyhal_timer_event_t event);

/*****
 * Global Variables
 *****/

bool timer_interrupt_flag = false;
bool led_blink_active_flag = true;

/* Variable for storing character read from terminal */
uint8_t uart_read_value;

/* Timer object used for blinking the LED */
cyhal_timer_t led_blink_timer;

/*****
 * Function Name: main
 *****/

* Summary:
* This is the main function. It sets up a timer to trigger a
* periodic interrupt. The main while loop checks for the status of a flag set
* by the interrupt and toggles an LED at 1Hz to create an LED blinky. The
* while loop also checks whether the 'Enter' key was pressed and
* stops/restarts LED blinking.
*
* Parameters:
* none
*
* Return:
* int
```

#### 4 Getting started with XMC7000 MCU design

```

*
*****/
int main(void)
{
    cy_rslt_t result;

    /* Initialize the device and board peripherals */
    result = cybsp_init();

    /* Board init failed. Stop program execution */
    if (result != CY_RSLT_SUCCESS)
    {
        CY_ASSERT(0);
    }

    /* Enable global interrupts */
    __enable_irq();

    /* Initialize retarget-io to use the debug UART port */
    result = cy_retarget_io_init(CYBSP_DEBUG_UART_TX, CYBSP_DEBUG_UART_RX,
                               CY_RETARGET_IO_BAUDRATE);

    /* retarget-io init failed. Stop program execution */
    if (result != CY_RSLT_SUCCESS)
    {
        CY_ASSERT(0);
    }

    /* Initialize the User LED */
    result = cyhal_gpio_init(CYBSP_USER_LED, CYHAL_GPIO_DIR_OUTPUT,
                             CYHAL_GPIO_DRIVE_STRONG, CYBSP_LED_STATE_OFF);

    /* GPIO init failed. Stop program execution */
    if (result != CY_RSLT_SUCCESS)
    {
        CY_ASSERT(0);
    }

    /* \x1b[2J\x1b[;H - ANSI ESC sequence for clear screen */
    printf("\x1b[2J\x1b[;H");

    printf("***** "
           "Hello World! Example "
           "***** \r\n\n");

    printf("Hello World!!!\r\n\n");

    printf("For more projects, "
           "visit our code examples repositories:\r\n\n");

    printf("https://github.com/Infineon/"
           "Code-Examples-for-ModusToolbox-Software\r\n\n");

```

#### 4 Getting started with XMC7000 MCU design

```

/* Initialize timer to toggle the LED */
timer_init();

printf("Press 'Enter' key to pause or "
      "resume blinking the user LED \r\n\r\n");

for (;;)
{
    /* Check if 'Enter' key was pressed */
    if (cyhal_uart_getc(&cy_retarget_io_uart_obj, &uart_read_value, 1)
        == CY_RSLT_SUCCESS)
    {
        if (uart_read_value == '\r')
        {
            /* Pause LED blinking by stopping the timer */
            if (led_blink_active_flag)
            {
                cyhal_timer_stop(&led_blink_timer);

                printf("LED blinking paused \r\n");
            }
            else /* Resume LED blinking by starting the timer */
            {
                cyhal_timer_start(&led_blink_timer);

                printf("LED blinking resumed\r\n");
            }

            /* Move cursor to previous line */
            printf("\x1b[1F");

            led_blink_active_flag ^= 1;
        }
    }

    /* Check if timer elapsed (interrupt fired) and toggle the LED */
    if (timer_interrupt_flag)
    {
        /* Clear the flag */
        timer_interrupt_flag = false;

        /* Invert the USER LED state */
        cyhal_gpio_toggle(CYBSP_USER_LED);
    }
}

/*****
 * Function Name: timer_init
 *****/

* Summary:
 * This function creates and configures a Timer object. The timer ticks

```

#### 4 Getting started with XMC7000 MCU design

```

* continuously and produces a periodic interrupt on every terminal count
* event. The period is defined by the 'period' and 'compare_value' of the
* timer configuration structure 'led_blink_timer_cfg'. Without any changes,
* this application is designed to produce an interrupt every 1 second.
*
* Parameters:
* none
*
*****/
void timer_init(void)
{
    cy_rslt_t result;

    const cyhal_timer_cfg_t led_blink_timer_cfg =
    {
        .compare_value = 0,           /* Timer compare value, not used */
        .period = LED_BLINK_TIMER_PERIOD, /* Defines the timer period */
        .direction = CYHAL_TIMER_DIR_UP, /* Timer counts up */
        .is_compare = false,          /* Don't use compare mode */
        .is_continuous = true,        /* Run timer indefinitely */
        .value = 0                    /* Initial value of counter */
    };

    /* Initialize the timer object. Does not use input pin ('pin' is NC) and
    * does not use a pre-configured clock source ('clk' is NULL). */
    result = cyhal_timer_init(&led_blink_timer, NC, NULL);

    /* timer init failed. Stop program execution */
    if (result != CY_RSLT_SUCCESS)
    {
        CY_ASSERT(0);
    }

    /* Configure timer period and operation mode such as count direction,
    duration */
    cyhal_timer_configure(&led_blink_timer, &led_blink_timer_cfg);

    /* Set the frequency of timer's clock source */
    cyhal_timer_set_frequency(&led_blink_timer, LED_BLINK_TIMER_CLOCK_HZ);

    /* Assign the ISR to execute on timer interrupt */
    cyhal_timer_register_callback(&led_blink_timer, isr_timer, NULL);

    /* Set the event on which timer interrupt occurs and enable it */
    cyhal_timer_enable_event(&led_blink_timer, CYHAL_TIMER_IRQ_TERMINAL_COUNT,
        7, true);

    /* Start the timer with the configured settings */
    cyhal_timer_start(&led_blink_timer);
}

/*****

```

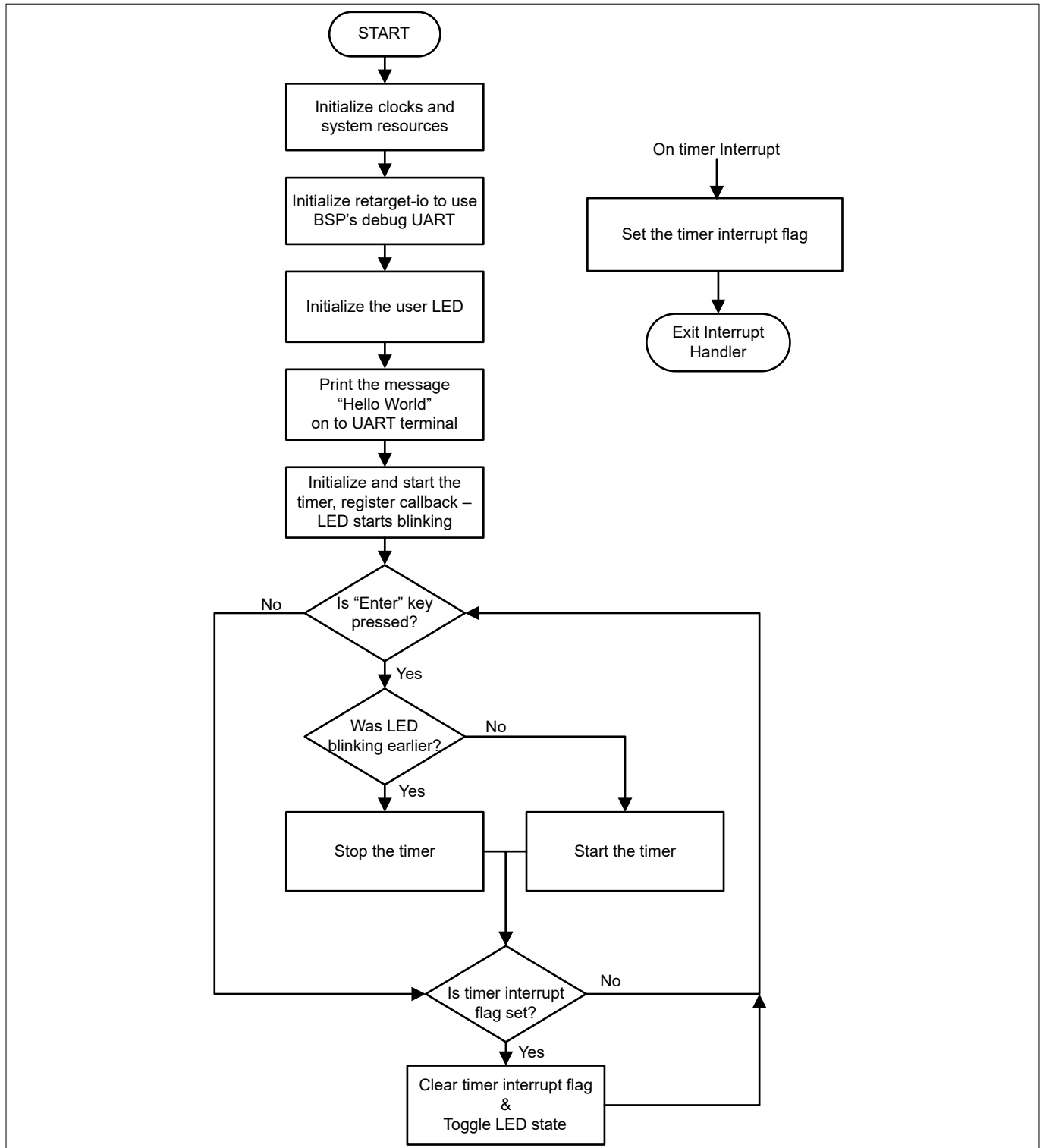


## 4 Getting started with XMC7000 MCU design

```
* Function Name: isr_timer
*****
* Summary:
* This is the interrupt handler function for the timer interrupt.
*
* Parameters:
*   callback_arg   Arguments passed to the interrupt callback
*   event           Timer/counter interrupt triggers
*
*****/
static void isr_timer(void *callback_arg, cyhal_timer_event_t event)
{
    (void) callback_arg;
    (void) event;

    /* Set the interrupt flag and process it from the main while(1) loop */
    timer_interrupt_flag = true;
}
```

#### 4 Getting started with XMC7000 MCU design



**Figure 20 Firmware flowchart**

This completes the summary of how the firmware works in the code example. Feel free to explore the source files for a deeper understanding.

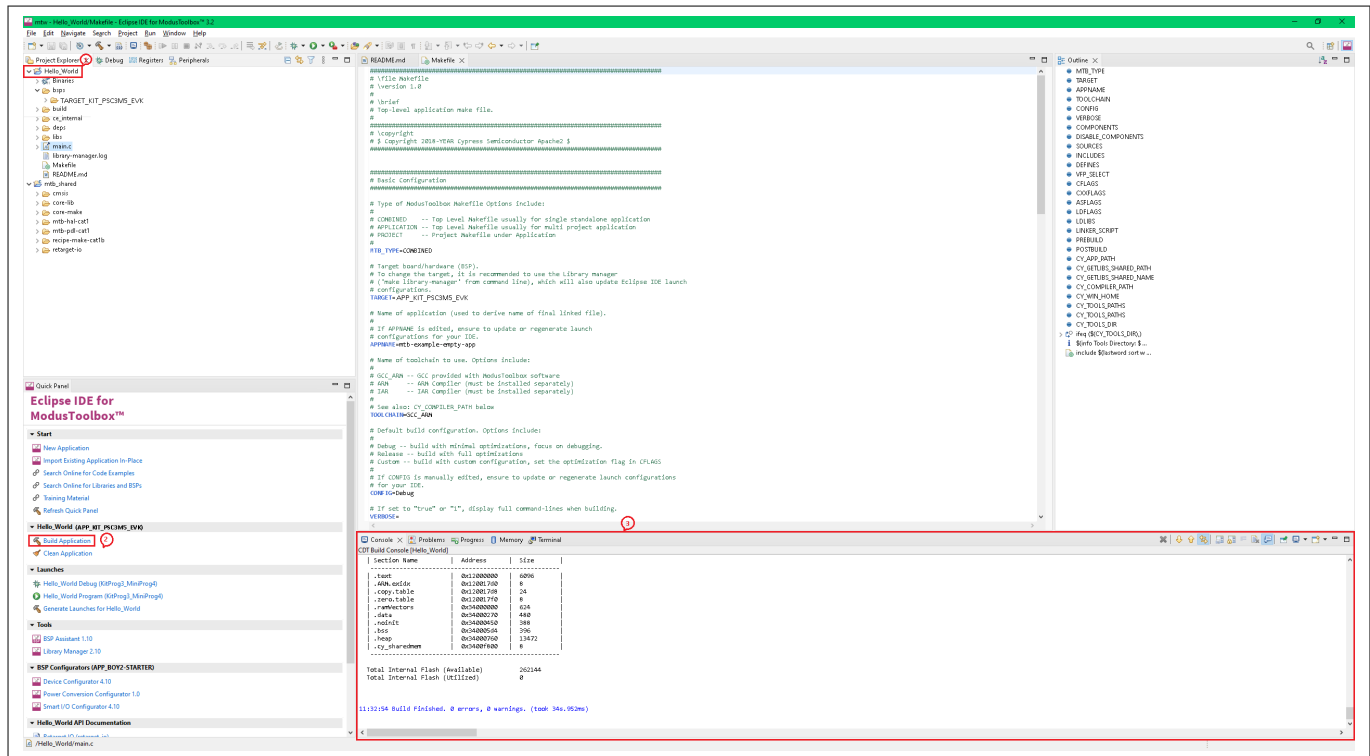
### 4.7 Build the application

This section shows how to build the application.

1. Select the application project in the **Project Explorer** view.

## 4 Getting started with XMC7000 MCU design

2. Click **Build Application** shortcut under the **<name>** group in the **Quick Panel**.  
It selects the build configuration from the Makefile and compiles/links all projects that constitute the application. By default, Debug configurations are selected.
3. The **Console view** lists the results of the build operation, as [Figure 21](#) shows.



**Figure 21** Build the application

If you encounter errors, revisit earlier steps to ensure that you completed all the required tasks.

**Note:** You can also use the command-line interface (CLI) to build the application. See the **Build system** section in the [ModusToolbox™ tools package user guide](#). This document is located in the `/docs_<version>/` folder in the ModusToolbox™ installation.

## 4.8 Program the device

This section shows how to program the XMC7000 MCU.

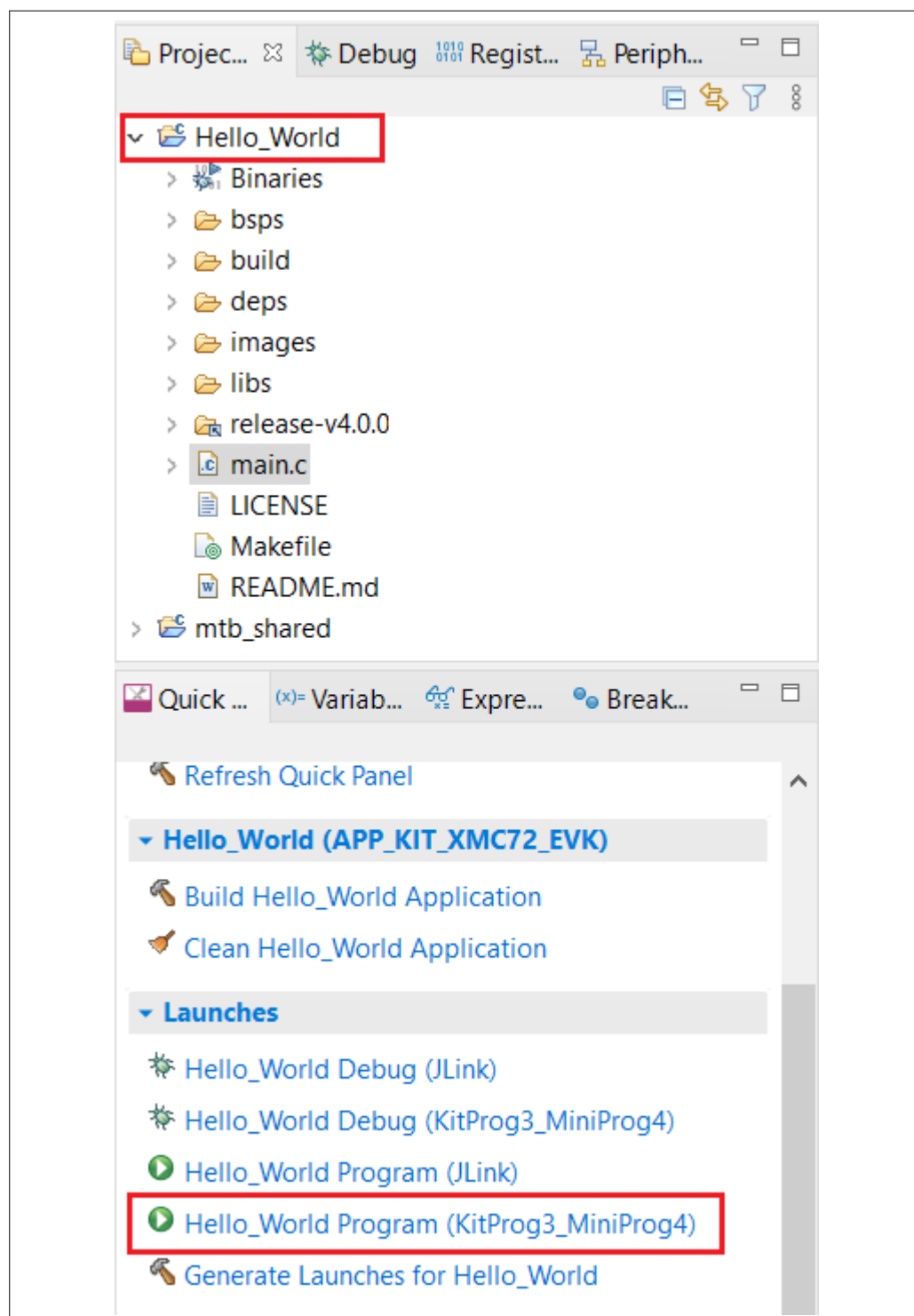
ModusToolbox™ software uses the OpenOCD protocol to program and debug applications on XMC7000 MCUs. For ModusToolbox™ software to identify the device on the kit, the kit must be running KitProg3. See the [Programming/debugging using Eclipse IDE](#) section for details.

If you are using a development kit with a built-in programmer, connect the board to your computer using the USB cable.

If you are developing on your own hardware, you may need a hardware programmer/debugger, for example, a [JLink](#) or [ULinkpro](#).

1. Program the application.
  - a. Connect to the board and perform the following.
  - b. Select the application project and click on the **<application name><name> Program (KitProg3\_MiniProg4)** shortcut under the **Launches** group in the **Quick Panel**, as [Figure 22](#) shows. The IDE will select and run the appropriate run configuration. Note that this step will also perform a build if any files have been modified since the last build.

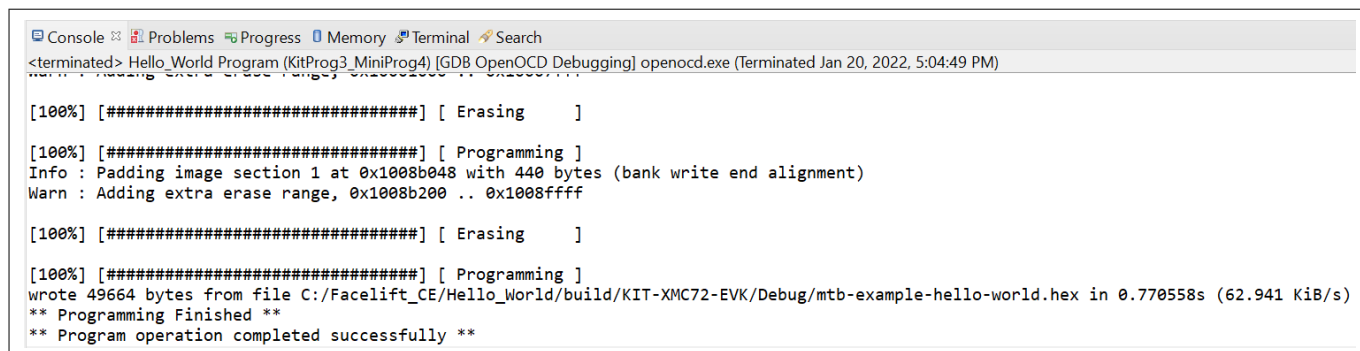
#### 4 Getting started with XMC7000 MCU design



**Figure 22** Programming an application to a device

The **Console** view lists the results of the programming operation, as [Figure 23](#) shows.

## 4 Getting started with XMC7000 MCU design



```

[100%] [#####] [ Erasing      ]
[100%] [#####] [ Programming ]
Info : Padding image section 1 at 0x1008b048 with 440 bytes (bank write end alignment)
Warn : Adding extra erase range, 0x1008b200 .. 0x1008ffff
[100%] [#####] [ Erasing      ]
[100%] [#####] [ Programming ]
wrote 49664 bytes from file C:/Facelift_CE/Hello_World/build/KIT-XMC72-EVK/Debug/mtb-example-hello-world.hex in 0.770558s (62.941 KiB/s)
** Programming Finished **
** Program operation completed successfully **
  
```

**Figure 23** Console – programming results

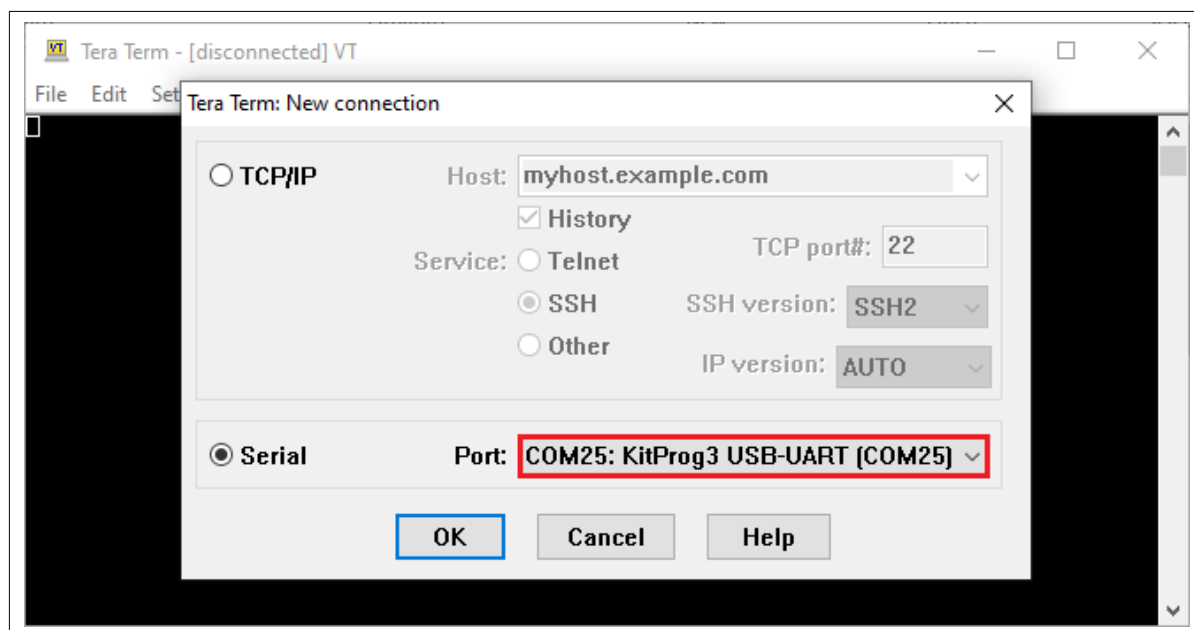
### 4.9 Test your design

This section describes how to test your design.

Follow these steps to observe the output of your design. This note uses Tera Term as the UART terminal emulator to view the results, but you can use any terminal of your choice to view the output.

#### 1. Select the serial port

Launch Tera Term and select the USB-UART COM port as [Figure 24](#) shows. Note that your COM port number may be different.

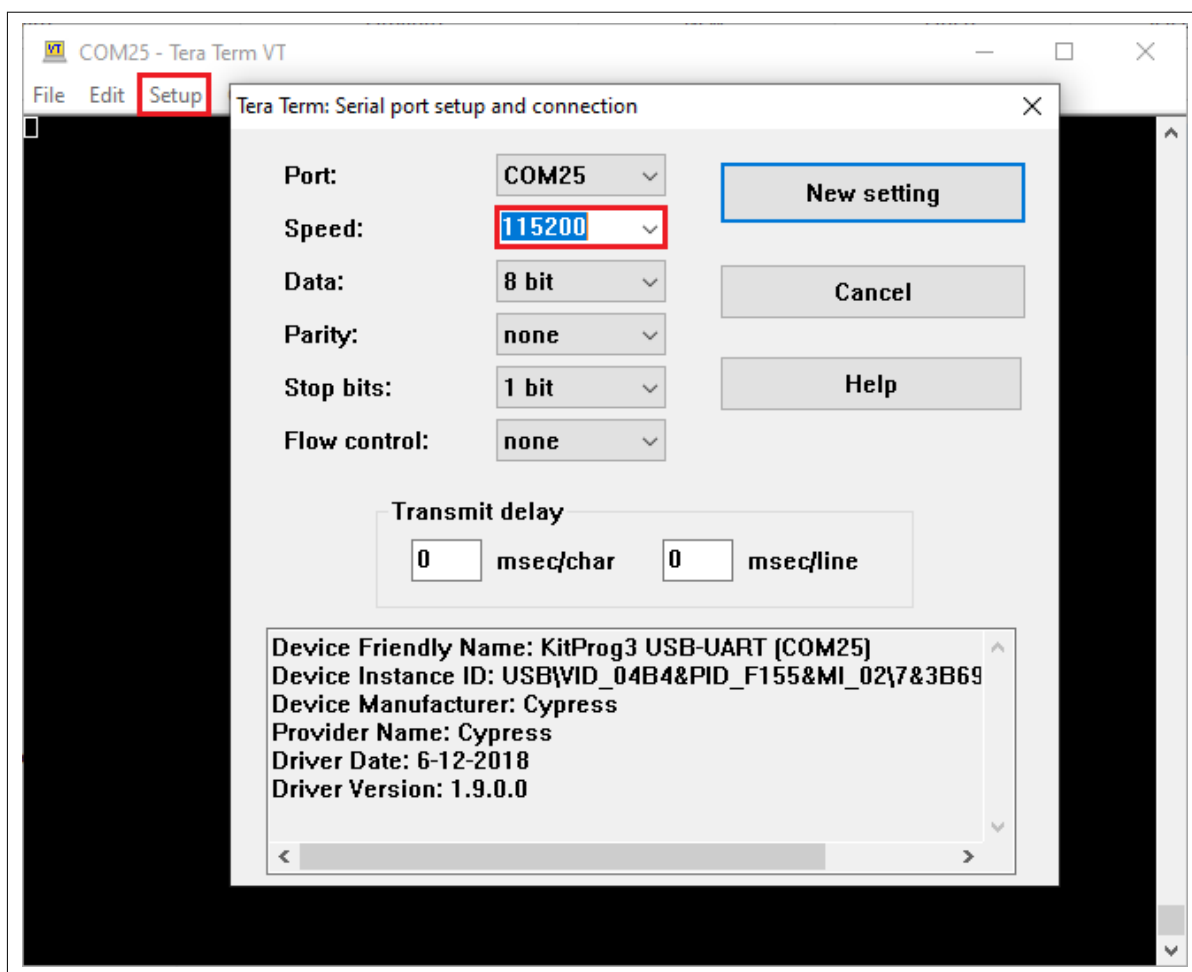


**Figure 24** Selecting the KitProg3 COM port in Tera Term

#### 2. Set the baud rate

Set the baud rate to 115200 under **Setup > Serial port** as [Figure 25](#) shows.

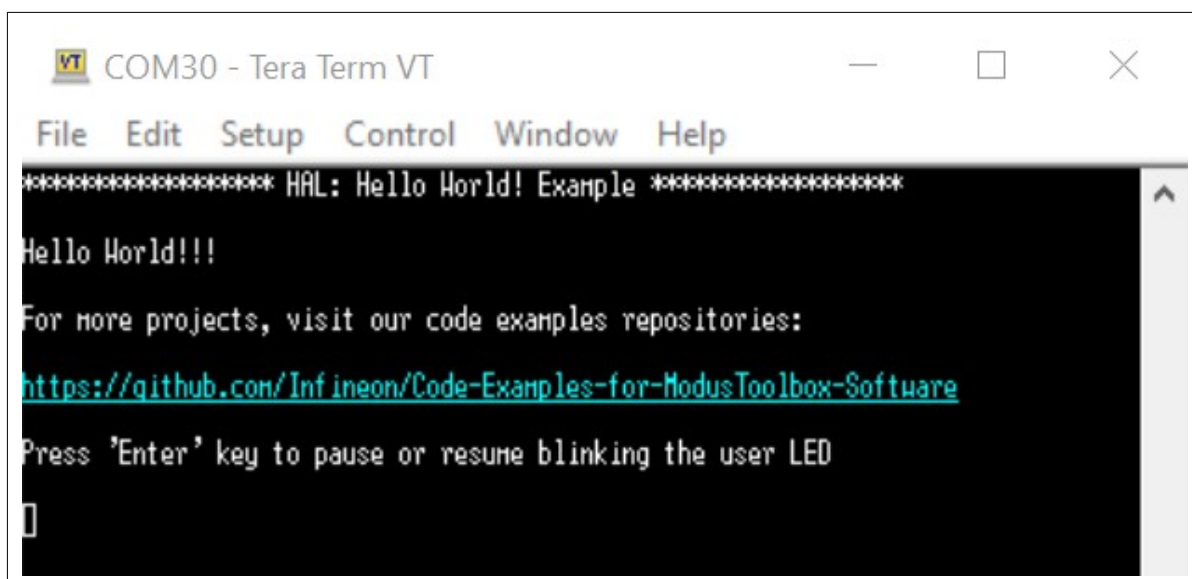
#### 4 Getting started with XMC7000 MCU design



**Figure 25** Configuring the baud rate in Tera Term

#### 3. Reset the device

Press the reset switch (**SW1**) on the kit. A message appears on the terminal as [Figure 26](#) shows. The user LED on the kit will start blinking.

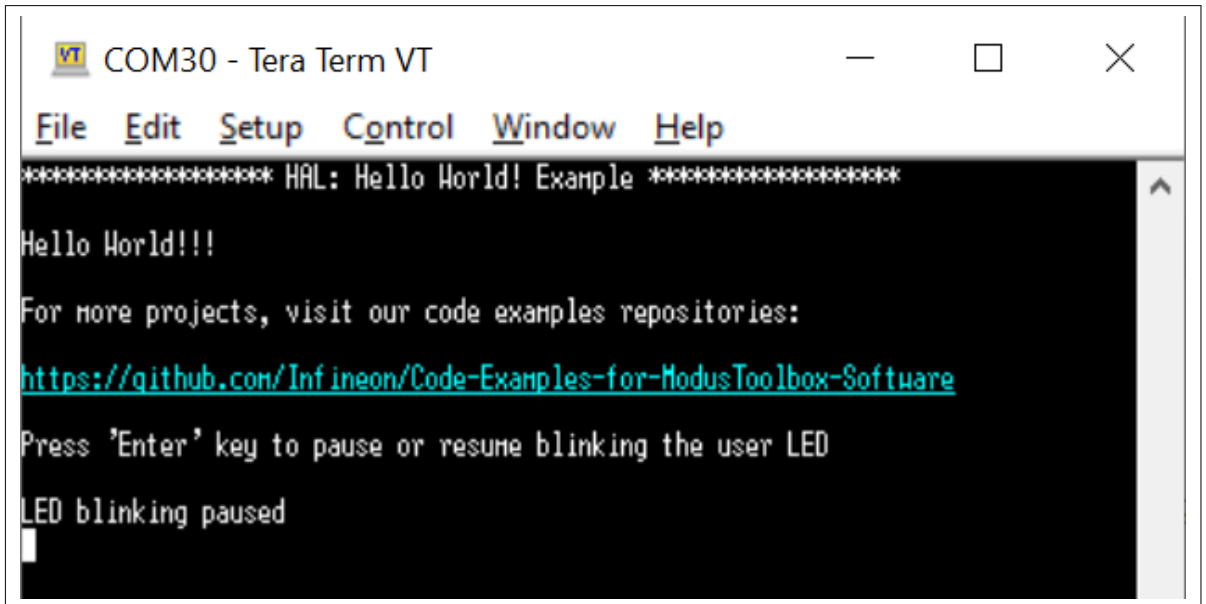


**Figure 26** Printed UART message

#### 4. Pause/resume LED blinking functionality

### 4 Getting started with XMC7000 MCU design

Press the **Enter** key to pause/resume blinking the LED. When the LED blinking is paused, a corresponding message will be displayed on the terminal as shown in [Figure 27](#).



```
COM30 - Tera Term VT
File Edit Setup Control Window Help
***** HAL: Hello World! Example *****
Hello World!!!
For more projects, visit our code examples repositories:
https://github.com/Infineon/Code-Examples-for-ModusToolbox-Software
Press 'Enter' key to pause or resume blinking the user LED
LED blinking paused
```

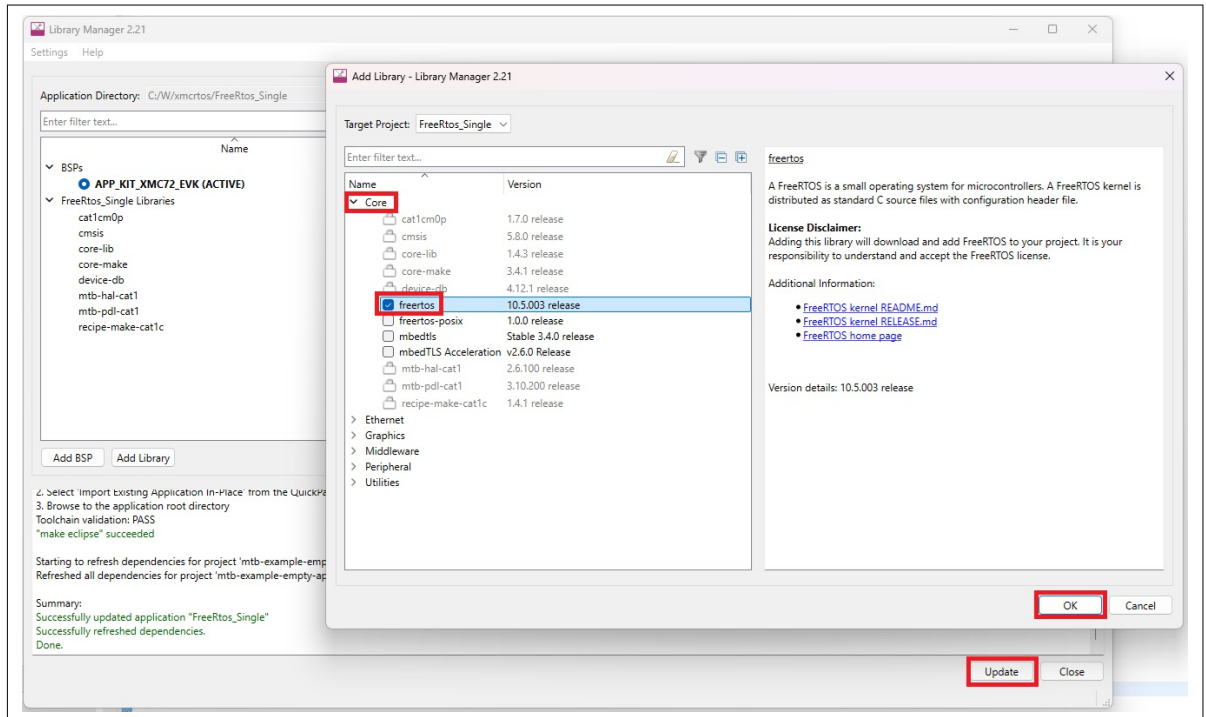
**Figure 27** Printed UART message

## 5 Adding FreeRTOS support for XMC7000

### 5 Adding FreeRTOS support for XMC7000

#### 5.1 For Single-core XMC7000 project

1. Create a new application as explained in [Create a new application](#).
2. Open **Library Manager** and select **freertos** library from 'Core'. Click on **Update** to add the library to the project.

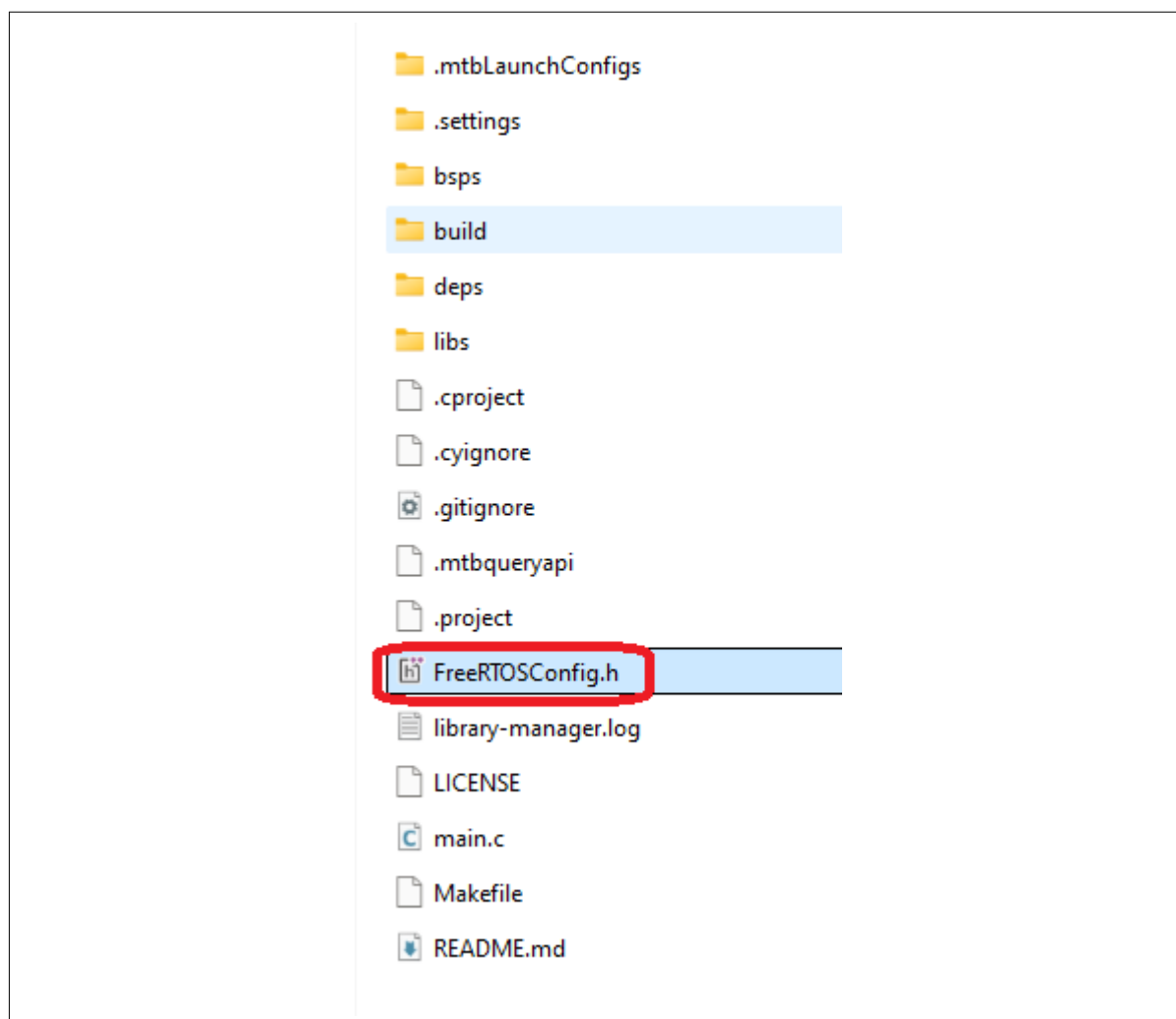


**Figure 28 Adding FreeRTOS library in a Single-core MTB project**

3. Add the below line to CE Makefile:  
COMPONENTS+=FREERTOS
4. Copy the FreeRTOSConfig.h from [https://github.com/Infineon/freertos/blob/master/Source/portable/COMPONENT\\_CM7/FreeRTOSConfig.h](https://github.com/Infineon/freertos/blob/master/Source/portable/COMPONENT_CM7/FreeRTOSConfig.h) to CE source files.



## 5 Adding FreeRTOS support for XMC7000



**Figure 29** FreeRTOSConfig.h in project structure

5. Open the copied *FreeRTOSConfig.h* file and remove the #warning This is a template. line.
6. Update the main.c using the below code snippet:

## 5 Adding FreeRTOS support for XMC7000

### Code listing 1: main.c file

```
#include "cyhal.h"
#include "cybsp.h"
#include "FreeRTOS.h"
#include "task.h"

#define LED_GPIO (CYBSP_USER_LED)

/*****
 * Macros
 *****/

/*****
 * Global Variables
 *****/

/*****
 * Function Prototypes
 *****/

/*****
 * Function Definitions
 *****/

/*****
 * Function Name: blinky
 *****/
* Summary:
* This is a FreeRTOS task used to toggle an LED.
*
* Parameters:
* void
*
* Return:
* int
*
*****/
void blinky(void * arg)
{
    (void)arg;

    /* Initialize the LED GPIO pin */
    cyhal_gpio_init(LED_GPIO, CYHAL_GPIO_DIR_OUTPUT,
                    CYHAL_GPIO_DRIVE_STRONG, CYBSP_LED_STATE_OFF);

    for(;;)
    {
        /* Toggle the LED periodically */
        cyhal_gpio_toggle(LED_GPIO);
    }
}
```

## 5 Adding FreeRTOS support for XMC7000

```

        vTaskDelay(500);
    }
}

/*****
 * Function Name: main
 *****/
* Summary:
* This function creates a FreeRTOS task.
*
* Parameters:
* void
*
* Return:
* int
*
*****/
int main(void)
{
    BaseType_t retval;
    cy_rslt_t result;

    /* Initialize the device and board peripherals */
    result = cybsp_init() ;
    if (result != CY_RSLT_SUCCESS)
    {
        CY_ASSERT(0);
    }

    __enable_irq();

    retval = xTaskCreate(blinky, "blinky", configMINIMAL_STACK_SIZE, NULL, 5, NULL);

    if (pdPASS == retval)
    {
        vTaskStartScheduler();
    }

    for (;;)
    {
        /* vTaskStartScheduler never returns */
    }
}

```

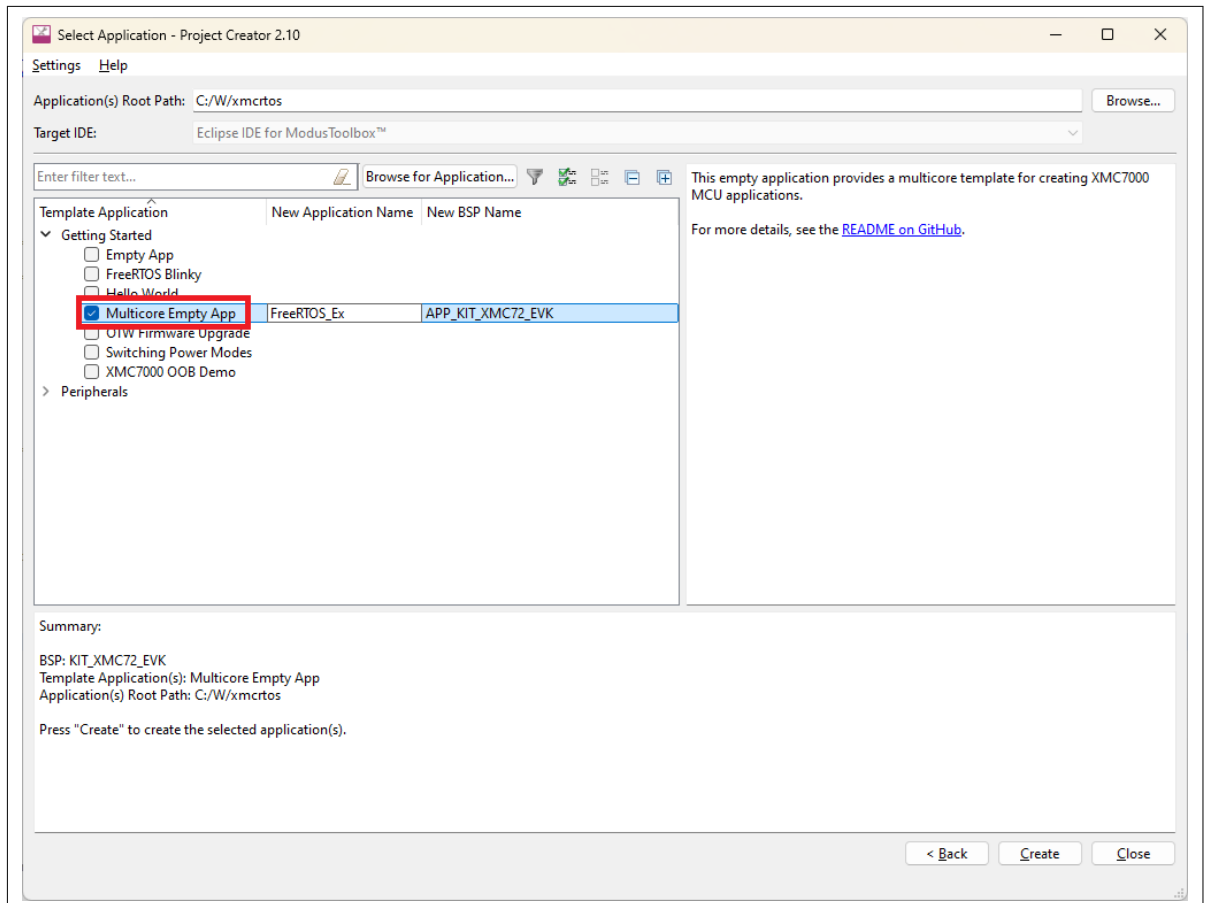
7. Build the project and program the KIT\_XMC72\_EVK. Refer to [Program the device](#) for more details on programming the board.
8. Observe that the LED on the XMC72 EVK starts blinking. The FreeRTOS component has created a task for blinking an LED and it task is getting triggered successfully.

## 5 Adding FreeRTOS support for XMC7000

### 5.2 For Multi-core project

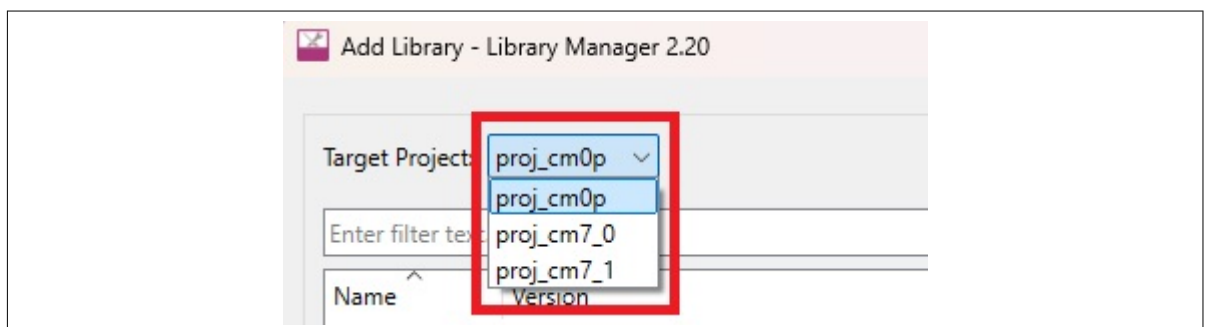
1. Create a multi-core project by selecting Multicore Empty App during project creation. Refer [Create a new application](#) for more details. Rename the project as required.

**Note:** *Empty App project creates a starter project for CM7\_0 core (single-core) and the remaining 2 cores are put to sleep at the start. Multicore Empty App creates a starter project with all 3 cores enabled. Multicore Empty App is a combination of 3 projects, proj\_cm0p, proj\_cm7\_0 and proj\_cm7\_1.*



**Figure 30 Create Multicore Empty App**

2. Open **Library Manager**. Select the **Target Project** from the drop-down list and add **freertos** library to all three projects. (proj\_cm0p, proj\_cm7\_0, proj\_cm7\_1) of the project. Click on **Update** to add the library to all 3 projects.



**Figure 31 Select Target Project in Library Manager**

5 Adding FreeRTOS support for XMC7000

Table 3

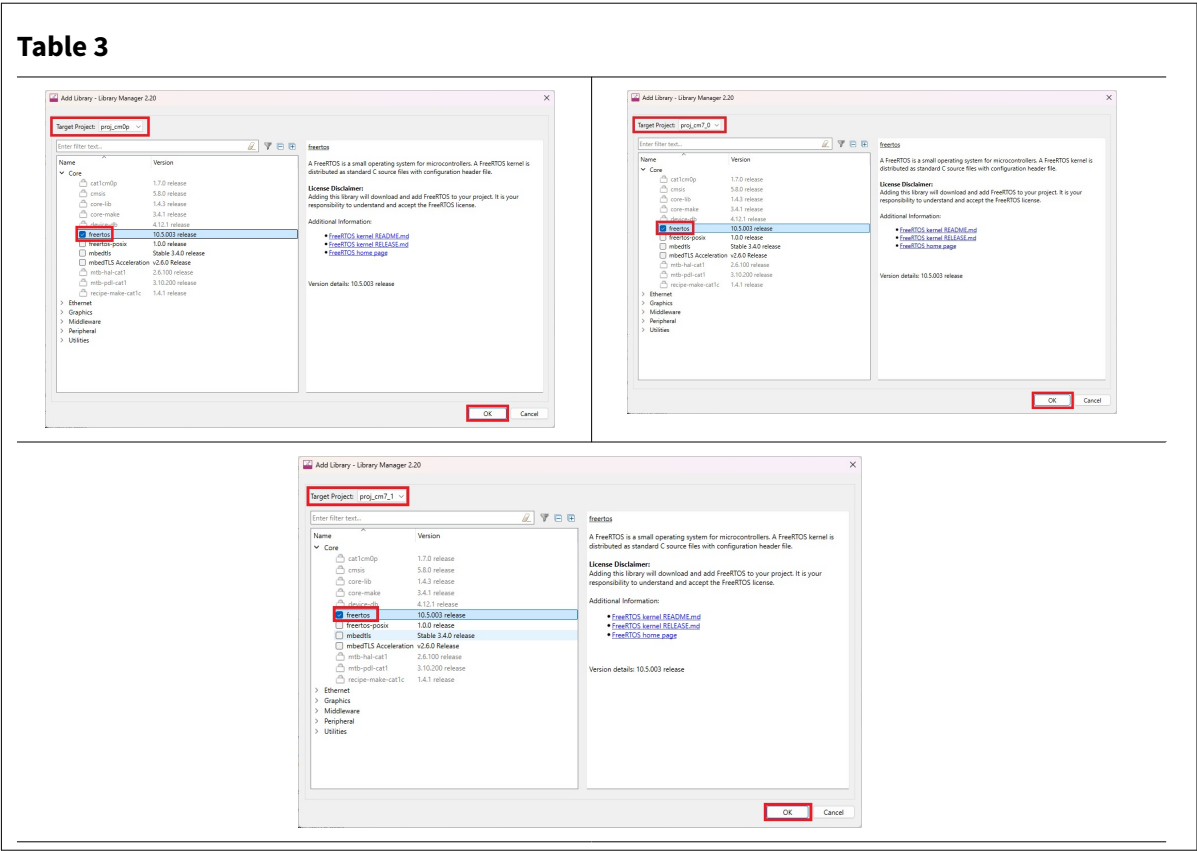
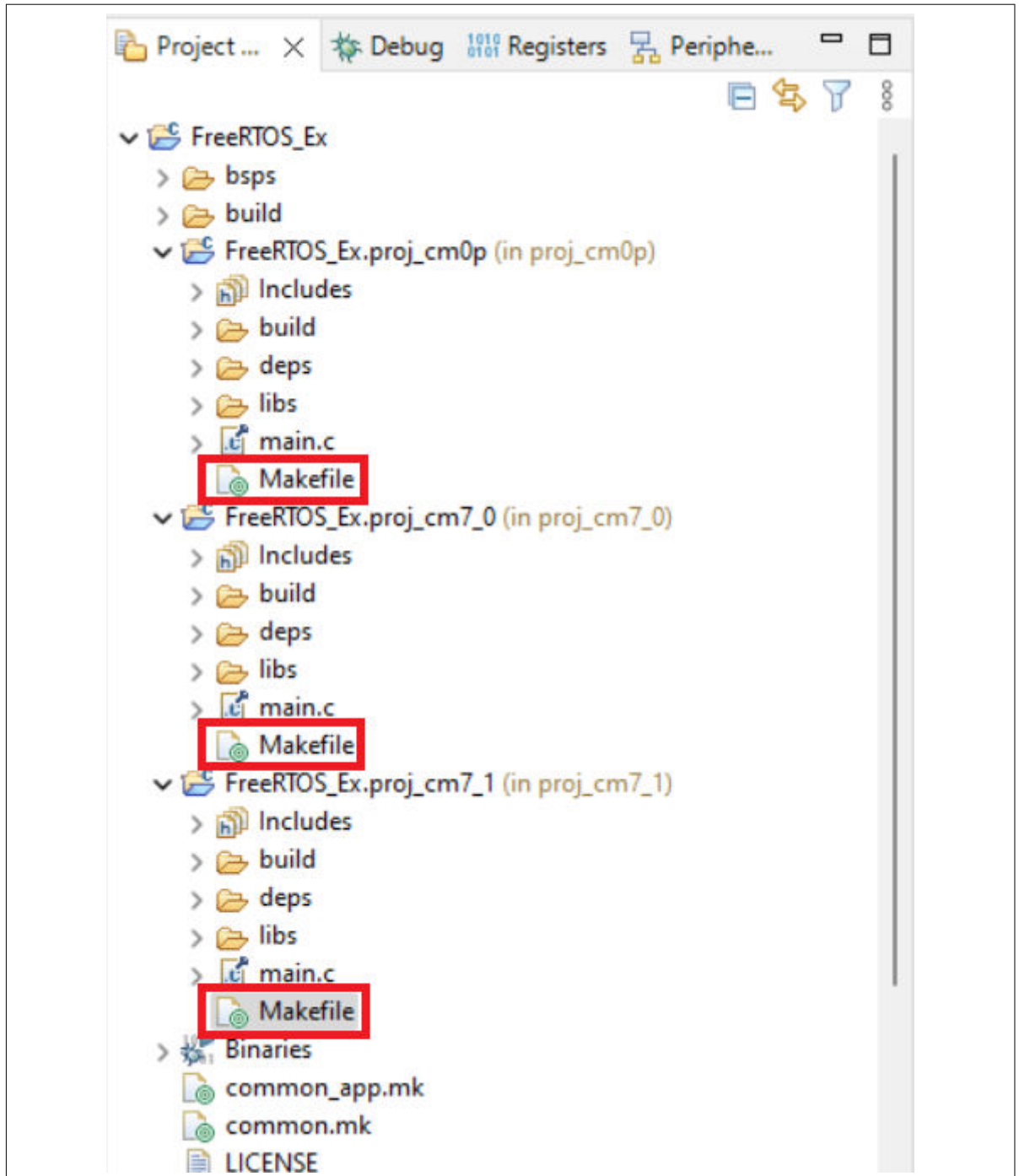


Figure 32 Adding FreeRTOS library to all three projects

3. Add the below line to the Makefile of all 3 projects:  
COMPONENTS+=FREERTOS

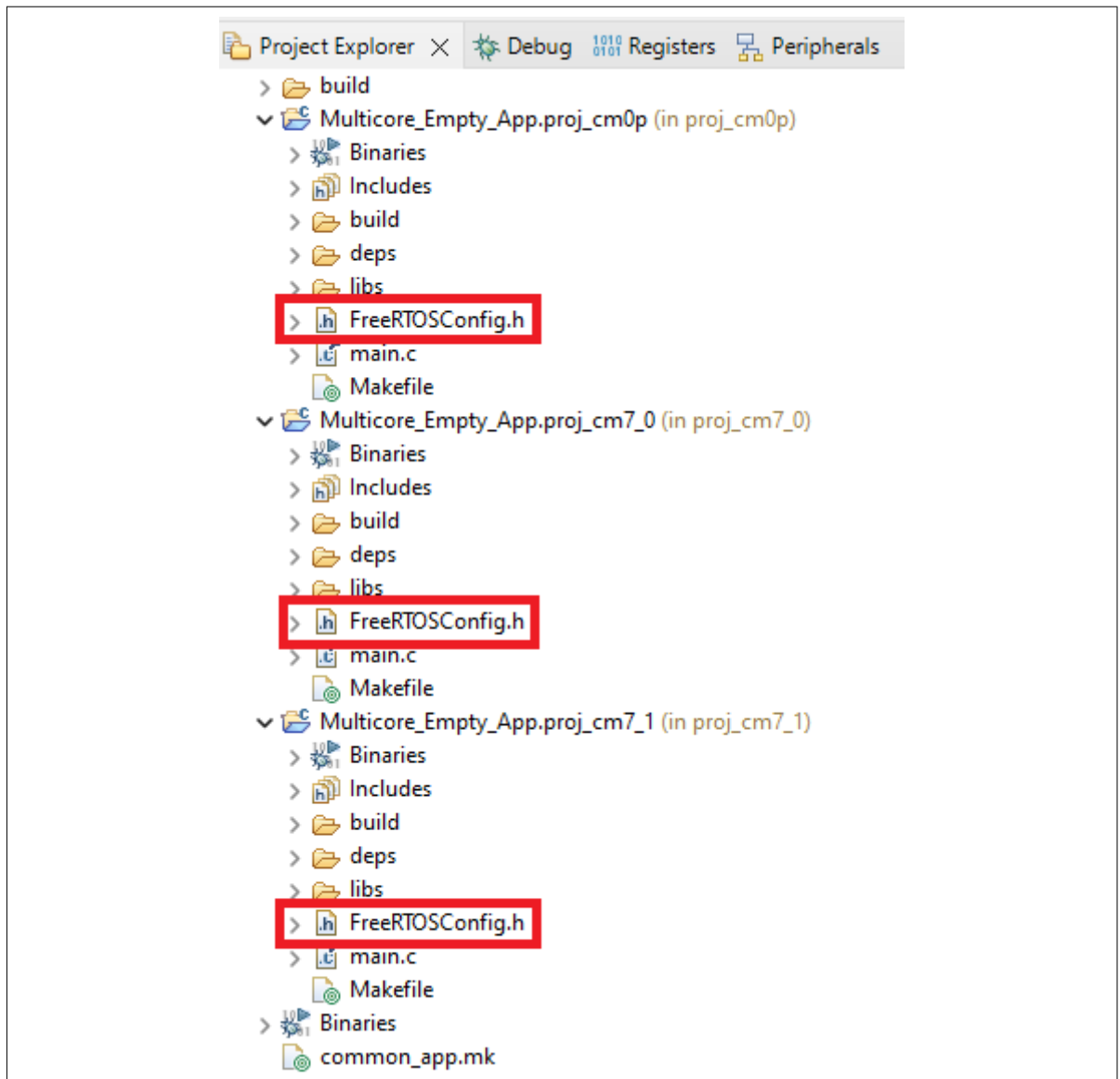
## 5 Adding FreeRTOS support for XMC7000



**Figure 33** Makefile of 3 projects

4. Copy the FreeRTOSConfig.h from [https://github.com/Infineon/freertos/blob/master/Source/portable/COMPONENT\\_CM0/FreeRTOSConfig.h](https://github.com/Infineon/freertos/blob/master/Source/portable/COMPONENT_CM0/FreeRTOSConfig.h) to proj\_cm0p project.
5. Copy the FreeRTOSConfig.h from [https://github.com/Infineon/freertos/blob/master/Source/portable/COMPONENT\\_CM7/FreeRTOSConfig.h](https://github.com/Infineon/freertos/blob/master/Source/portable/COMPONENT_CM7/FreeRTOSConfig.h) to proj\_cm7\_0 and proj\_cm7\_1 projects.

## 5 Adding FreeRTOS support for XMC7000



**Figure 34** Project structure after adding FreeRTOSConfig.h

6. Open the copied *FreeRTOSConfig.h* file and remove the `#warning This is a template.` line. Ensure to do so, in all three *FreeRTOSConfig.h* files.
7. The FreeRTOS library requires MCWDT to periodically trigger the tasks. MCWDT0, MCWDT1 and MCWDT2 are assigned to be used by the CM0P, CM7\_0 and CM7\_1 cores respectively.  
In a multi-core project, the HAL Hardware manager is not natively aware of other cores, so we call `cyhal_hwmgr_reserve` to inform it of resources (MCWDT) that will be used elsewhere. The `main.c` of CM0P reserves the MCWDT1 and MCWDT2 in hardware manager so that, CM0P uses only MCWDT0 for FreeRTOS timing requirements. Similarly, CM7\_0 reserves MCWDT0 and MCWDT2 resources and CM7\_1 reserves MCWDT0 and MCWDT1 resources.
8. Replace the `main.c` of `proj_cm0p` with code snippet given below:

## 5 Adding FreeRTOS support for XMC7000

### Code listing 2: main.c file (proj\_cm0p)

```
#include "cyhal.h"
#include "cybsp.h"
#include "FreeRTOS.h"
#include "task.h"

#define LED_GPIO (CYBSP_USER_LED1)

/*****
 * Macros
 *****/

#define CM7_DUAL 1

/*****
 * Global Variables
 *****/

const cyhal_resource_inst_t mcwdt0 =
{ .type = CYHAL_RSC_LPTIMER, .block_num = 0U, .channel_num = 0U, }
;
const cyhal_resource_inst_t mcwdt1 =
{ .type = CYHAL_RSC_LPTIMER, .block_num = 1U, .channel_num = 0U, }
;
const cyhal_resource_inst_t mcwdt2 =
{ .type = CYHAL_RSC_LPTIMER, .block_num = 2U, .channel_num = 0U, }
;

/*****
 * Function Prototypes
 *****/

/*****
 * Function Definitions
 *****/

/*****
 * Function Name: blinky
 *****/

* Summary:
* This is a FreeRTOS task that is used to blink an LED.
*
* Parameters:
* void
*
* Return:
* int
*
*****/

void blinky(void * arg)
{
    (void)arg;

    /* Initialize the LED GPIO pin */
}
```



## 5 Adding FreeRTOS support for XMC7000

```

    cyhal_gpio_init(LED_GPIO, CYHAL_GPIO_DIR_OUTPUT,
                    CYHAL_GPIO_DRIVE_STRONG, CYBSP_LED_STATE_OFF);

    for(;;)
    {
        /* Toggle the LED periodically */
        cyhal_gpio_toggle(LED_GPIO);
        vTaskDelay(500);
    }
}

/*****
* Function Name: main
*****/
* Summary:
* This function creates a FreeRTOS task.
*
* Parameters:
* void
*
* Return:
* int
*
*****/
int main(void)
{
    BaseType_t retval;
    cy_rslt_t result;

    /* Initialize the device and board peripherals */
    result = cybsp_init() ;
    if (result != CY_RSLT_SUCCESS)
    {
        CY_ASSERT(0);
    }

    // Leave mcwdt0 available

    cyhal_hwmgr_reserve(&mcwdt1);

    cyhal_hwmgr_reserve(&mcwdt2);

    __enable_irq();

    /* Enable CM7_0/1. CY_CORTEX_M7_APPL_ADDR is calculated in linker script, check it in
    case of problems. */
    Cy_SysEnableCM7(CORE_CM7_0, CY_CORTEX_M7_0_APPL_ADDR);

    #if CM7_DUAL
        Cy_SysEnableCM7(CORE_CM7_1, CY_CORTEX_M7_1_APPL_ADDR);
    #endif /* CM7_DUAL */

```

### 5 Adding FreeRTOS support for XMC7000

---

```
retval = xTaskCreate(blinky, "blinky", configMINIMAL_STACK_SIZE, NULL, 5, NULL);

if (pdPASS == retval)
{
    vTaskStartScheduler();
}

for(;;)
{
}
}
```

9. Replace the main.c of proj\_cm7\_0 project with code snippet given below:

## 5 Adding FreeRTOS support for XMC7000

### Code listing 3: main.c file (proj\_cm7\_0)

```
#include "cyhal.h"
#include "cybsp.h"
#include "FreeRTOS.h"
#include "task.h"

#define LED_GPIO (CYBSP_USER_LED2)

/*****
 * Macros
 *****/

/*****
 * Global Variables
 *****/
const cyhal_resource_inst_t mcwdt0 =

{ .type = CYHAL_RSC_LPTIMER, .block_num = 0U, .channel_num = 0U, }
;

const cyhal_resource_inst_t mcwdt1 =

{ .type = CYHAL_RSC_LPTIMER, .block_num = 1U, .channel_num = 0U, }
;

const cyhal_resource_inst_t mcwdt2 =

{ .type = CYHAL_RSC_LPTIMER, .block_num = 2U, .channel_num = 0U, }
;

/*****
 * Function Prototypes
 *****/

/*****
 * Function Definitions
 *****/
/*****
 * Function Name: blinky
 *****/
* Summary:
* This is a FreeRTOS task that is used to blink an LED.
*
* Parameters:
* void
*
* Return:
* int
*
*****/
```

## 5 Adding FreeRTOS support for XMC7000

```

void blinky1(void * arg)
{
    (void)arg;

    /* Initialize the LED GPIO pin */
    cyhal_gpio_init(LED_GPIO, CYHAL_GPIO_DIR_OUTPUT,
                    CYHAL_GPIO_DRIVE_STRONG, CYBSP_LED_STATE_OFF);

    for(;;)
    {
        /* Toggle the LED periodically */
        cyhal_gpio_toggle(LED_GPIO);
        vTaskDelay(500);
    }
}

/*****
 * Function Name: main
 *****/
* Summary:
* This function creates a FreeRTOS task.
*
* Parameters:
* void
*
* Return:
* int
*
*****/
int main(void)
{
    BaseType_t retval;
    cy_rslt_t result;

    /* Initialize the device and board peripherals */
    result = cybsp_init() ;
    if (result != CY_RSLT_SUCCESS)
    {
        CY_ASSERT(0);
    }

    // leave mcwdt1 available

    cyhal_hwmgr_reserve(&mcwdt0);

    cyhal_hwmgr_reserve(&mcwdt2);

    __enable_irq();

    retval = xTaskCreate(blinky1, "blinky1", configMINIMAL_STACK_SIZE, NULL, 5, NULL);

    if (pdPASS == retval)
    {

```

### 5 Adding FreeRTOS support for XMC7000

---

```
    vTaskStartScheduler();  
}  
  
for (;;)   
{  
    /* vTaskStartScheduler never returns */  
}  
}
```

- 10.** Replace the main.c of proj\_cm7\_1 with code snippet given below:

## 5 Adding FreeRTOS support for XMC7000

### Code listing 4: main.c file (proj\_cm7\_1)

```
#include "cyhal.h"
#include "cybsp.h"
#include "FreeRTOS.h"
#include "task.h"

#define LED_GPIO (CYBSP_USER_LED3)

/*****
 * Macros
 *****/

/*****
 * Global Variables
 *****/

const cyhal_resource_inst_t mcwdt0 =
{ .type = CYHAL_RSC_LPTIMER, .block_num = 0U, .channel_num = 0U, }
;

const cyhal_resource_inst_t mcwdt1 =
{ .type = CYHAL_RSC_LPTIMER, .block_num = 1U, .channel_num = 0U, }
;

const cyhal_resource_inst_t mcwdt2 =
{ .type = CYHAL_RSC_LPTIMER, .block_num = 2U, .channel_num = 0U, }
;

/*****
 * Function Prototypes
 *****/

/*****
 * Function Definitions
 *****/

/*****
 * Function Name: blinky
 *****/

* Summary:
* This is a FreeRTOS task that is used to blink an LED.
*
* Parameters:
* void
*
* Return:
* int
*
*****/

void blinky2(void * arg)
{
    (void)arg;
}
```

## 5 Adding FreeRTOS support for XMC7000

```

/* Initialize the LED GPIO pin */
cyhal_gpio_init(LED_GPIO, CYHAL_GPIO_DIR_OUTPUT,
                 CYHAL_GPIO_DRIVE_STRONG, CYBSP_LED_STATE_OFF);

for(;;)
{
    /* Toggle the LED periodically */
    cyhal_gpio_toggle(LED_GPIO);
    vTaskDelay(500);
}
}

/*****
* Function Name: main
*****/
* Summary:
* This function creates a FreeRTOS task.
*
* Parameters:
* void
*
* Return:
* int
*
*****/
int main(void)
{
    BaseType_t retval;
    cy_rslt_t result;

    /* Initialize the device and board peripherals */
    result = cybsp_init() ;
    if (result != CY_RSLT_SUCCESS)
    {
        CY_ASSERT(0);
    }

    // leave mcwdt2 available

    cyhal_hwmgr_reserve(&mcwdt0);

    cyhal_hwmgr_reserve(&mcwdt1);

    __enable_irq();

    retval = xTaskCreate(blinky2, "blinky2", configMINIMAL_STACK_SIZE, NULL, 5, NULL);

    if (pdPASS == retval)
    {
        vTaskStartScheduler();
    }
}

```

## 5 Adding FreeRTOS support for XMC7000

```
for (;;)
{
    /* vTaskStartScheduler never returns */
}
}
```

11. Build the project and program the KIT\_XMC72\_EVK. Refer to [Program the device](#) for more details on programming the board.
12. Observe that all three LEDs on the XMC72 EVK starts blinking. The FreeRTOS component has created one task in each core. The LEDs are being handled within these tasks. All 3 tasks are getting triggered successfully.

**Note:** *The above code snippet is based on KIT\_XMC72\_EVK. The same procedure can be used to enable FreeRTOS on KIT\_XMC71\_EVK\_V1 BSP as well. However, KIT\_XMC71\_EVK\_V1 BSP has only 2 User LEDs. The above code example can be updated to include a UART output for one of the cores to observe the FreeRTOS functionality.*



---

## 6 Summary

### 6 Summary

This application note explored the XMC7000 MCU device architecture and the associated development tools. XMC7000 MCU is a truly programmable embedded system-on-chip with configurable analog and digital peripheral functions, memory, and a dual-core system on a single chip. The integrated features, embedded security, and low-power modes make XMC7000 MCU an ideal choice for smart home, IoT gateways, and other related applications.

## 7 References

## 7 References

For a complete and updated list of XMC7000 MCU code examples, please visit our [GitHub](#). For more XMC7000 MCU-related documents, please visit our [XMC7000 MCU](#) product web page.

[Table 4](#) lists the system-level and general application notes that are recommended for the next steps in learning about XMC7000 MCU and ModusToolbox™.

**Table 4 General and system-level application notes**

Document	Document name
<a href="#">AN234224</a>	Hardware design guide for the XMC7000 family
<a href="#">AN225588</a>	Using ModusToolbox™ software with a third-party IDE
<a href="#">AN234021</a>	Low power mode procedure in XMC7000 family

[Table 5](#) lists the application notes (AN) for specific peripherals and applications.

**Table 5 Documents related to XMC7000 MCU features**

Document	Document name
<b>System resources, CPU, and interrupts</b>	
<a href="#">AN234226</a>	How to use interrupt in the XMC7000 family
<a href="#">AN234225</a>	How to use DMA in the XMC7000 family
<b>Peripherals</b>	
<a href="#">AN234127</a>	How to use SCB in XMC7000 family
<a href="#">AN234380</a>	How to use ethernet controller in XMC7000 family
<a href="#">AN234227</a>	Using the SMIF in the XMC700 family
<a href="#">AN234197</a>	How to use trigger multiplexer in XMC700 family
<a href="#">AN234022</a>	How to use CAN FD in XMC7000 family
<a href="#">AN234119</a>	Timer, Counter and PWM (TCPWM) usage in XMC7000 family
<b>Device datasheet</b>	
<a href="#">002-33896</a>	XMC7100 datasheet
<a href="#">002-33522</a>	XMC7200 datasheet

---

## Glossary

## Glossary

This section lists the most commonly used terms that you might encounter while working with XMC™ family of devices.

- **Board support package (BSP):** A BSP is the layer of firmware containing board-specific drivers and other functions. The board support package is a set of libraries that provide firmware APIs to initialize the board and provide access to board level peripherals.
- **Hardware Abstraction Layer (HAL):** The HAL wraps the lower level drivers (like [MTB-PDL-CAT1](#)) and provides a high-level interface to the MCU. The interface is abstracted to work on any MCU.
- **KitProg:** The KitProg is an onboard programmer/debugger with USB-I2C and USB-to-UART bridge functionality. The KitProg is integrated onto most XMC™ development kits.
- **MiniProg3/MiniProg4:** Programming hardware for development that is used to program XMC™ devices on your custom board or XMC™ development kits that do not support a built-in programmer.
- **Personality:** A personality expresses the configurability of a resource for a functionality. For example, the SCB resource can be configured to be an UART, SPI, or I2C personalities.
- **Middleware:** Middleware is a set of firmware modules that provide specific capabilities to an application. Some middleware may provide network protocols (e.g. MQTT), and some may provide high level software interfaces to device features (e.g. USB, audio).
- **ModusToolbox™:** An Eclipse-based embedded design platform for embedded systems designers that provides a single, coherent, and familiar design experience, combining the industry's most deployed Wi-Fi and Bluetooth technologies, and the lowest power, most flexible MCUs with best-in-class sensing.
- **Peripheral Driver Library:** The peripheral driver library (PDL) simplifies software development for the XMC7000 MCU architecture. The PDL reduces the need to understand register usage and bit structures, thus easing software development for the extensive set of peripherals available.

---

**Revision history****Revision history**

Document version	Date of release	Description of changes
**	2022-01-25	Initial release.
*A	2022-06-17	Updated <a href="#">Figure 8</a> , <a href="#">Figure 9</a> , <a href="#">Figure 14</a> , <a href="#">Figure 15</a> , <a href="#">Figure 17</a> , <a href="#">Figure 18</a> , <a href="#">Figure 19</a> , <a href="#">Figure 21</a> , and <a href="#">Figure 22</a> .
*B	2023-01-26	Updated <a href="#">Section 4.5</a> according to MTB#8265.
*C	2023-05-24	Template update. Updated <a href="#">Getting started with XMC7000 MCU design</a> section.
*D	2023-06-09	Updated content with the latest release of ModusToolbox™ 3.1
*E	2023-08-29	Added note in <a href="#">Getting started with XMC7000 MCU design</a> section and updated other issues throughout the document.
*F	2024-12-16	Updated <a href="#">Firmware flow</a> , <a href="#">ModusToolbox™ applications</a> , <a href="#">Write firmware</a> , and <a href="#">Create a new application</a> . Added <a href="#">Adding FreeRTOS support for XMC7000</a> and <a href="#">Figure 16</a> . Updated <a href="#">Figure 14</a> and <a href="#">Figure 15</a> .

## Trademarks

All referenced product or service names and trademarks are the property of their respective owners.

**Edition 2024-12-16**

**Published by**

**Infineon Technologies AG**  
**81726 Munich, Germany**

**© 2024 Infineon Technologies AG**  
**All Rights Reserved.**

**Do you have a question about any aspect of this document?**

**Email: [erratum@infineon.com](mailto:erratum@infineon.com)**

**Document reference**  
**IFX-ckr1674573825368**

## Important notice

The information contained in this application note is given as a hint for the implementation of the product only and shall in no event be regarded as a description or warranty of a certain functionality, condition or quality of the product. Before implementation of the product, the recipient of this application note must verify any function and other technical information given herein in the real application. Infineon Technologies hereby disclaims any and all warranties and liabilities of any kind (including without limitation warranties of non-infringement of intellectual property rights of any third party) with respect to any and all information given in this application note.

The data contained in this document is exclusively intended for technically trained staff. It is the responsibility of customer's technical departments to evaluate the suitability of the product for the intended application and the completeness of the product information given in this document with respect to such application.

## Warnings

Due to technical requirements products may contain dangerous substances. For information on the types in question please contact your nearest Infineon Technologies office.

Except as otherwise explicitly approved by Infineon Technologies in a written document signed by authorized representatives of Infineon Technologies, Infineon Technologies' products may not be used in any applications where a failure of the product or any consequences of the use thereof can reasonably be expected to result in personal injury.